AN INTELLIGENT, ROBUST APPROACH TO VOLUMETRIC AIRCRAFT SIZING

A Dissertation
Presented to the Academic Faculty

By

Eric Upton

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in Aerospace Engineering

Georgia Institute of Technology

August, 2007

AN INTELLIGENT, ROBUST APPROACH TO VOLUMETRIC AIRCRAFT SIZING

Approved by:

Dr. Dimitri Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Daniel Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. James Craig
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Danielle Soban
School of Aerospace Engineering
Georgia Institute of Technology

Robert McKinley
Director of Engineering
DRS Unmanned Technologies, inc.

Date Approved: 7 May 2007

It is only when we become conscious of our part in life, however modest, that we shall be happy. Only then will we be able to live in peace and die in peace, for only this lends meaning to life and to death.

The man who can see the miraculous in a poem, who can take pure joy from music, who can break his bread with comrades, opens his window to the same refreshing wind off the sea. He too learns a language of men.

But too many men are left unawakened.

– Antoine De Saint Exupéry.

Acknowledgements

The author wishes to thank the following for their help with this project:

The members of the thesis committee, for their time and guidance.

John Hayes, for programming assistance and a willingness to listen to yet another syntax problem.

Rob McDonald, for assistance and advice with the GTS library.

Carolyn Upton and Reid Thomas, who read it even though they didn't have to.

And to Yvette Upton, who read it more than anyone, provided chocolate and diet soda for long days and late nights, and who risked bodily harm walking the dogs so this work could be as strong as possible.

Table of Contents

List of Tables

List of Figures

List of Acronyms

3D	Three Dimensional
ASDL	Aerospace Systems Design Laboratory at The Georgia Institute of Technology
BB	Branch and Bound
BVE	Boxed Volume Equivalent
CAD	Computer Aided Design
CG	Center of Gravity
DoE	Design of Experiments
GA	Genetic Algorithm
GLIB	Graphics Library (an Open Source Library of C functions)
GTRI	Georgia Tech Research Institute
GTS	GNU Triangulated Surface Library
GSL	GNU Scientific Library
HALE	High Altitude, Long Endurance
LCC	Life-Cycle Cost
LTA	Lighter Than Air
LVE	Liquid Volume Equivalent
NAVAIR	Naval Air Systems Command
NGP	Next Generation Piston
NP	Nondeterministic Polynomial
PLM	Product Lifecycle Management
SA	Simulated Annealing

SVE	Simplified Volume Equivalent
UAV	Uninhabited Aerial Vehicle
URETI	University Research Engineering and Technology Institutes

Summary

Advances in computational power have produced great strides in the later design and production portions of an aircraft's life cycle, and these advances have included the internal layout component of the design and manufacturing process. However, conceptual and preliminary design tools for internal layout remain primarily based on historical regressions and estimations – a situation that becomes untenable when considering revolutionary designs or component technologies. An assessment of the state-of-the-art techniques for volumetric considerations used in current aircraft design literature highlights the disadvantages of most current approaches and demonstrates the need for an improved method.

Bringing internal layout information forward in the design process can encourage the same level of benefits enjoyed by other disciplines as advances in aerodynamics, structures and other fields propagate forward in the design of complex systems. Accurate prediction of the volume required to contain all of an aircraft's internal components results in a more accurate prediction of aircraft specifications, mission effectiveness, and costs, helping determine if an aircraft is the best choice for continued development.

This is not a computationally simple problem, however, and great care must be taken to ensure the efficiency of any proposed solution by making wise choices in terms of search techniques that can intelligently and rapidly find a valid solution if one exists. Any solution must also address the uncertainty inherent in describing internal components early in the design process in order to

produce a robust solution. Implementing a methodology that applies notions of an intelligent search for a solution, as well as deals robustly with component sizing, produces a high chance of success.

Development of an effective, rapid method for assessing the volumetric characteristics of an aircraft in the context of the conceptual and preliminary design processes can offer many of the benefits of a complete internal layout without the assignment of resources typical in the detail phase of the design process. A simplified volumetrically sizing methodology for aircraft is presented here that is designed to be integrated with later phases of the design process.

A prototype tool for demonstrating a volumetric sizing methodology is discussed and demonstrated in a limited capacity using a combination of original code and publicly available libraries. The usable parameters of the code are explained, and recommendations are made toward future volumetric tools and their integration with existing and proposed sizing techniques.

The underlying volumetric sizing methodology is demonstrated using a combination of original code and readily accessible design tools. The demonstration platform is the design of a notional UAV, where internal volume considerations drive changes in fuselage configuration in order to most appropriately meet customer needs. The UAV design is presented with an array of options for satisfying the customer needs, as well as strategies to ensure that volumetric data is preserved as the design process continues. Consequences of design choices are presented in terms of aircraft performance and cost, demonstrating how designers can use volumetric awareness to enhance both the economic and flight performance of an aircraft design.

1 Introduction and Background

Historically, aircraft sizing techniques have concentrated on weight as the primary metric of value. Aircraft designs have often been iterated through a set of performance and economic analyses until the lightest aircraft that can complete the mission set is created. In most conventional cases, this focus has been a sensible one, as aircraft have always been constrained by a desire for low weight and efficient structural design. For revolutionary designs or those that are densely packed, however, traditional weight-based sizing approaches may not be capable of completely addressing the aircraft's sizing needs.

Lowering aircraft weight has been historically seen as the simplest way to reduce aircraft cost. A lighter aircraft has, in theory, fewer components (or smaller ones) and requires less material. Among aircraft with similar technology levels, weight is proportional to acquisition cost while being inversely proportional to performance, so minimizing weight has a beneficial effect on both performance and cost. For some designs, however, minimizing weights, whether they are empty weights or gross weights, is insufficient. At some point, the cost of fabricating a lighter part will outweigh its performance advantage.

Modern designs also tend to be driven by multiple desires beyond simple requirements for performance or acquisition cost. Aircraft sized for a minimum weight may not offer the lowest operating cost, for example, or they may not be as well suited to an expanding set of missions. They may also require tooling or production techniques that are impractical or expensive to implement. The

materials used in construction may also contribute to a design conflict, such as the need to balance the weight benefit of a proposed new technology with the extra cost and uncertainty inherent in its development. Sizing during the conceptual design phase exclusively on weight can also have unexpected side effects. The level of fidelity typically used during conceptual design is insufficient to fully describe the aircraft – the implementation of simple regressions through historical data or empirical relations for materials and structural strength, for example, may not be sufficient.

The result of this may be an aircraft that simply cannot meet all of its design criteria. The aircraft may not fly as fast or as far as originally envisioned, or the aircraft may not have sufficient space for its cargo and all of its internal components. These issues can all lead to expensive redesign late in the design process or a final aircraft that does not meet all of its design requirements.

Historically, the internal arrangement of aircraft has been completed using methods that begin with a set of assumptions about how the internal arrangement will work. For relatively simple evolutionary designs, or for those designs that intrinsically have a good deal of surplus space, this is not a problem. However, for revolutionary designs, or for designs whose requirements may change suddenly, this approach has serious flaws.

In the early days of commercial jet transportation, the Lockheed Jetstar was one of the first attempts to bring jet performance to the general aviation market, and it was considered quite a technological achievement. Unfortunately, extremely high fuel consumption from its four turbojet engines required additional "slipper tanks" beneath each wing, as there was not sufficient room inside the aircraft for the required fuel. These tanks were fitted in the wings, and

their large size, easily visible in Figure 1, almost dominates the aircraft's wing; the tanks created additional drag on the aircraft, affecting its performance[1].

Figure 1: *Lockheed Jetstar at the USAF Museum in Dayton, OH*[2]

Similar issues plagued the Concorde, which required extensive redesign late in its development after being plagued with internal equipment issues that proved to be more political than technical[3]. This further delayed its entry to service, contributing to its failure in the marketplace.

More modern computer aided design (CAD) techniques have helped advance the design process through integration of large amounts of data into a single, easily manipulated environment. Modern CAD packages can fully define an aircraft from initial component selection through virtual manufacturing. Unfortunately, these processes are still far too time consuming for use during conceptual design, with the time required to fully describe a modern, new design being typically measured in years. The Joint Strike Fighter (JSF) program, begun

in 1993 as a response to the cancellation of two other programs, did not result in a flyable prototype aircraft until 2000[4]. In 2001 the Lockheed Martin F-35 was selected as the winner of the JSF competition, though the aircraft did not fly in production form until late in 2006[5]. Similarly, the initial design of the F-22 was begun in the mid-1980s, yet the first prototype did not fly until 1997[6].

Implementation of CAD, then, has not proven to be the simple solution to accelerating aircraft development. In the case of both the F-35 and F-22, the aircraft being considered are highly sophisticated, densely packaged, and technically advanced. These types of aircraft could be enhanced most by a process that helps determine if an aircraft has sufficient room inside for all of its cargo, crew, and payload. The F-22's stealthy configuration and the F-35's lift fan, for example, demonstrate technologies that are not easy to accommodate using traditional tools. Instead of simply exercising traditional tools, designers on these aircraft were forced to find ways of dealing with aircraft that did not fit together in the way traditional aircraft had. Particularly in the case of the lift fan in vertical takeoff versions of the F-35, these aircraft required a rework of traditional rules of component placement and layout.

A cutaway drawing of the F-22 is shown in Figure 2, in comparison to a cutaway of one of Lockheed's earlier designs, the Vega. While these two planes performed different roles, they do present a good example of the state of aircraft design within the same company over a seventy-year period. Even at first glance, the empty space in the Vega's wings and tail are obvious, as is the empty space behind the rear seat. Those spaces are nearly non-existent in the F-22, with only the tail having a modicum of space available, as the F-22's wings are efficiently filled with fuel.

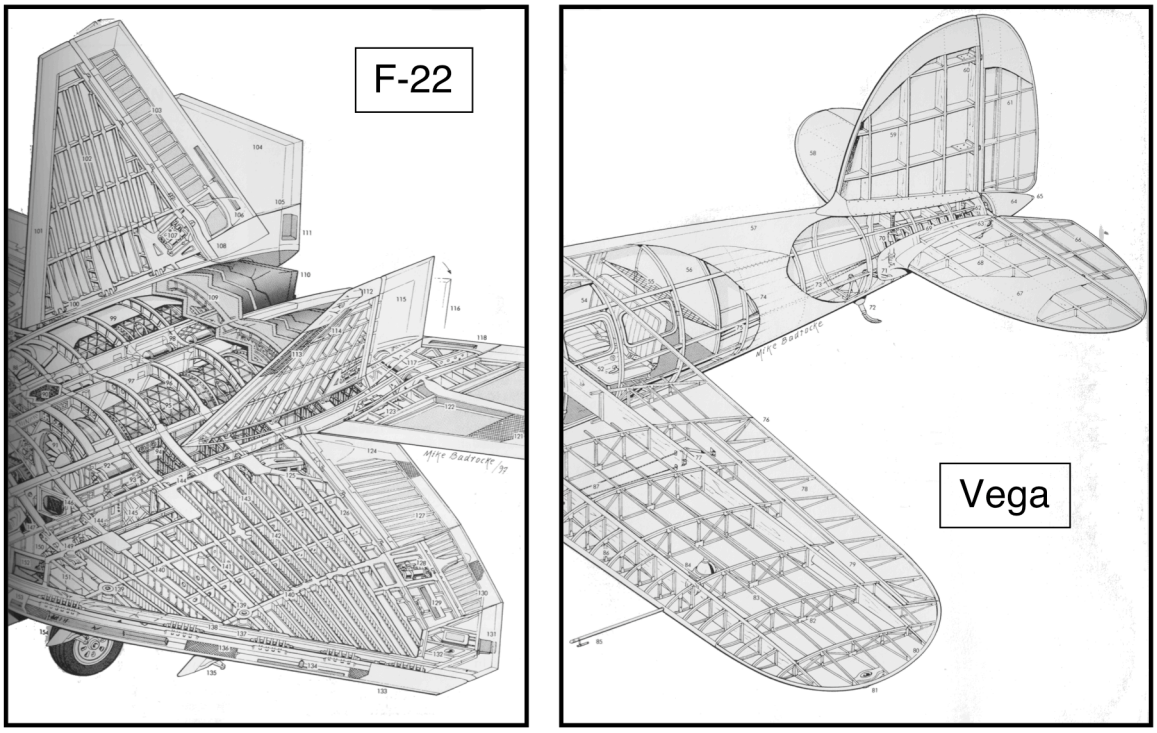


Figure 2: *Lockheed F-22 and Vega cutaways*[7]

Modern design tools have evolved to allow designers to assess the aerodynamic, structural, and propulsion effects of advanced technologies, but what about the inside of the aircraft? What are the consequences to the internal arrangements of these aircraft when applying these technologies, and what tools are required to help predict them effectively?

For aircraft too different or complex to be fully described by traditional methods, a continuum of problems can develop from insufficiently detailed considerations of the components inside the airframe. At one extreme, an aircraft sized simply to the minimum weight or some other simplified standard may not have sufficient space onboard for the components required for its missions,

despite having the power necessary to lift them. This problem with internal volume is not unheard of. It is most likely to impact design of technologically sophisticated aircraft, such as supersonic aircraft and fighters, where aerodynamic, performance, and structural requirements contrast with the need of the aircraft to carry payload and fuel. It is particularly common in aircraft that are refitted for a new mission after production has been completed.

Certain proposed new designs for high altitude/long endurance (HALE) aircraft, micro-sized autonomous vehicles, and fuel cell-powered general aviation aircraft also share this design quandary. The next generation of fuel cell-based propulsion systems have a low volumetric power density compared to current reciprocating and turbine systems. These relatively larger engines will result in a large potential for volume problems with these vehicles. Increasing loads of sensors and other unconventional payloads have the same effect, as do the particular requirements of micro-sized vehicles. A cutaway of a notional hybrid UAV HALE design created for NAVAIR is shown in Figure 3[8]. This aircraft makes use of a solid oxide fuel cell as well as a compression-ignition reciprocating engine to achieve long range and endurance. Volumetric uncertainty about the space required for fuel reformation and other fuel cell accessories cast doubt on the final configuration and generated what is an aerodynamically disadvantaged fuselage configuration[9].

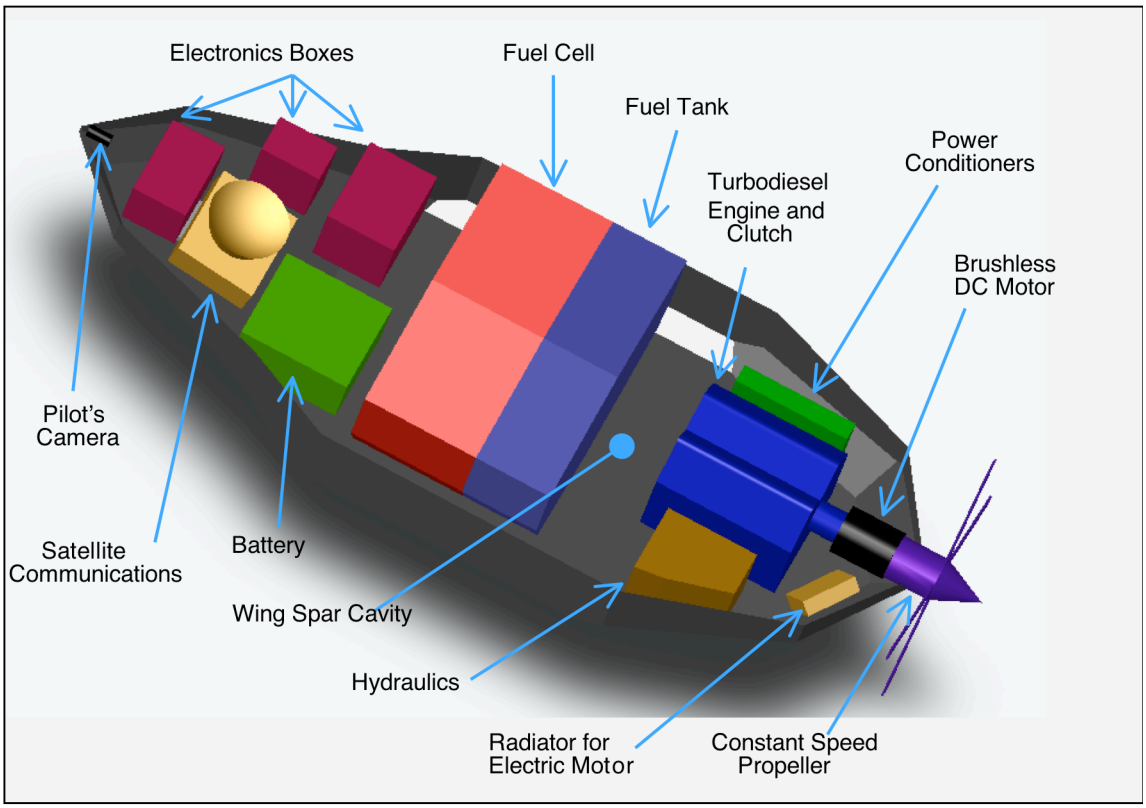


Figure 3: *Internal layout of a notional fuel cell UAV*

At the other extreme are aircraft that are "over-designed" with respect to volume. An aircraft design with more room than necessary is an aircraft that will be heavier than necessary, have worse aerodynamic performance, or both. Traditional, evolutionary designs can fall into this category in the event of significant and unexpected developments in technologies that reduce the size of internal components. Modern general aviation aircraft have only recently developed in a way that handles this issue. The tapered fuselages of aircraft like Cessna's next generation piston (NGP) prototype aircraft contrast greatly with older designs, such as Cessna's own 172 Skyhawk, a design that first flew in

1954[10,11]. The prototype NGP is shown in Figure 4, with a current-model 172 in Figure 5. A look inside each plane will reveal the large amount of empty space in the 172 behind the cabin. The recently designed NGP removes this wasted space while improving performance. The solution to avoiding aircraft designs that have too much space, as well as to those that have too little, is to involve volume considerations from the beginning of the sizing and synthesis development.

Figure 4: *Cessna NGP at Airventure Oskosh 2006*[12]

Figure 5: *Current-production Cessna 172*[13]

1.1 Development of the Aircraft Design Paradigm

Aircraft design has kept pace with developments in a variety of technically challenging fields. Advances in materials science allowed airplanes to transition from wood and fabric construction, to steel tubes, to aluminum, and to advanced composites. Propulsion system changes led to the jet age and later advances in efficiency. Similarly, modern electronics are revolutionizing navigation through advances in cockpit instrumentation and use of the Global Positioning System.

The designs which may have been sketched in chalk on the floors of factory hangars a hundred years ago are now completely described in digital form before the first piece of metal or composite is formed[14]. Collaborative design efforts spanning continents electronically in mere moments and decentralized production have replaced the rooms full of draftsmen with French curves sitting just above the production floor and the rooms full of women figuring aerodynamic calculations with pencils and slide rules.

The path of technological development in aircraft design has an important tradition in the aerospace community. In all of this development, a hierarchy of aerospace disciplines formed at the logical intersection of practical engineering, available analytical technologies, and the development of theory within government, academia and industry. It also produced what have become the fundamental disciplines of aerodynamics, propulsion, structures, and controls. It is the consolidation of the individual characteristics of all of these disciplines that make modern aircraft possible.

Partially responsible for the early success of the Wright Brothers, for example, was their unwavering insistence on rigorous experimentation and the successful integration of all aspects of airplane design, particularly control of the aircraft, which other designers of the time considered something of an afterthought[15]. Rather than developing a design based on what "seemed" right, the Wrights were able to demonstrate the effectiveness of their ideas before committing them to wood, fabric, and metal. Relying on data from their wind tunnel and their own experiences as bicycle builders and mechanics, the Wrights were able to see the actual application of their ideas about flight.

The decision to rely on experimentation rather than guesswork marked an early delineation in the development of an aircraft design paradigm. For the first time in aircraft design, a sound foundation supported ideas on how to achieve flight. Beginning with aerodynamics, experimentation spread quickly to the rest of what became the fundamental disciplines of aeronautics. It then spread across other aspects of aircraft development into any space where improvement could be found. Basic studies of wing performance, for example, grew into the design and application of high lift devices. Some early controls issues focused on standardizing the cockpit as various aircraft manufacturers for a while each had their own methods of steering the craft[16].

Unfortunately, experimentation is as expensive as it is effective, and aircraft designers were always in search of more effective design tools. In an attempt to avoid building a new prototype every time something didn't fit quite right or fly quite as fast as expected, designers went "back to the books". With the surge in interest in aviation in the early parts of the twentieth century, and with a host of developments from related fields such as thermodynamics and

fluid dynamics, a large base of relevant equations was developed. With only a few sets of corrections needed to allow accurate predictions of reality, equations were developed to mathematically describe characteristics such as the flow of air over wings, the structural performance of components, and the amount of fuel required for a mission. Modified versions of many of these equations, such as Prandtl's lifting-line theory, developed during World War I, continue to be used today[17].

The basic assumptions behind these equations meant, however, that the results may not be appropriate for all conditions, and the rapid advances in aviation-related technologies led to a number of serious problems related to these early equations. Early considerations of supersonic flight, for example, were complicated by predictions of an infinitely large drag rise as velocity approached the speed of sound[18]. This was not a problem when aircraft flew slowly, but as the Second World War approached, aircraft were beginning to encroach on this new performance limit. Similarly, the Second World War marked the first practical attempt at transporting large amounts of personnel and materiel across long distances by air. Likewise, the ideas that would lead to the jet age were being implemented for the first time. Empirical data was being accumulated that would allow the tweaking of those basic sets of equations and aid in the creation of new equations that the Wrights and their early contemporaries could have only dreamed about.

The large volumes of empirical results also meant that regressions could be used to predict characteristics of new aircraft models with amazing accuracy – as long as the aircraft remained within the bounds of the existing data points. Designs outside the existing ranges brought with them sometimes-disastrous

problems as prediction curves that worked extremely well within one range of data points failed to be accurate outside of that range.

Accurately predicting characteristics of aircraft outside of known ranges requires a new set of tools. Some of the fundamental formulas from the early part of the twentieth century are still accurate when used in their most general cases, but the amount of computational effort required to solve them in their general forms was far beyond that available when they were written, and the simplified forms used too many assumptions that were no longer valid with the coming of high-performance aircraft in the days after the Second World War. Then, with the advent of practical computers in the 1950s and 1960s, the ability to rapidly compute complex problems became feasible. Advances in computational power since then have supported a corresponding rise in the complexity of calculations done in the course of an aircraft's design. Relatively rapid numerical solutions to challenging design problems allowed improved fidelity over a range of previously unpredictable solutions and fostered a further evolution of the ever-shifting aircraft design paradigm.

Through the development of aircraft design, certain disciplines have always led the way. Aerodynamic studies were always among the first to be influenced by changes in design methods. Propulsion and structures also rapidly took advantage of the improved capabilities available with the new computational resources. Internal layout, however, historically lagged behind the other disciplines.

Traditionally, the only requirements for internal layout were that all of the internal components fit the space available, none of the arrangements violated common-sense assumptions (i.e., the landing gear must be at the bottom of the

aircraft), and the entire arrangement must be structurally sound and within the weight-and-balance range of the aircraft. Most of these concerns have been addressed easily through empirical relations and engineering judgment, particularly for designs that have significant amounts of surplus interior space.

A quest for increased performance and revolutionary changes in propulsion systems for many aircraft has led to more densely packed airframes and more difficulty finding room for interior components. A desire for these advanced designs within the aerospace community has made designing aircraft more complicated, and this desire has also reduced the effectiveness of traditional design techniques.

1.2 Traditional Sizing Methodologies

Aircraft designers have a wide variety of design techniques to call upon for the synthesis of their designs. Many of these techniques focus on a relatively narrow portion of the design process, and none really describes the process completely. As the work here focuses on what is commonly considered the conceptual and early preliminary phases of aircraft design, methodologies that focus on these areas will be of most importance. A graphical look at the three traditional phases of aircraft design is shown in Figure 6.

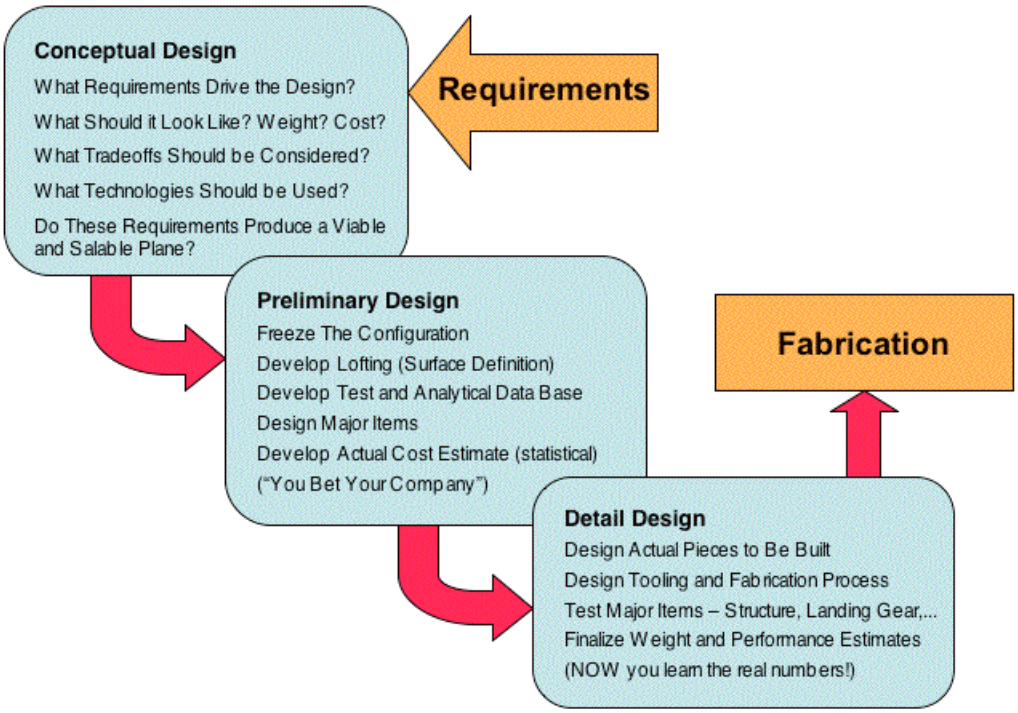


Figure 6: *The Phases of the Design Process (after Raymer)*[19]

The conceptual design phase is where most of the "big" decisions are made. The first choice is the definition of requirements. What is this aircraft going to carry? What will it do with this load? Where is it going to go? How long will it take to get there? Then the basic configuration: is the aircraft an airplane or a helicopter? a blimp? maybe a rocket? Once the fundamental configuration is chosen, the synthesis of the aircraft begins in earnest. Does it have wings? How many? Where do they go? How about engines? What kind? Where do you put the gas? Does it even use gas? How fast does it need to go? How far? At the end of this process, the designer is left with the basics of an aircraft. This is one step past the "napkin phase", where the entire design can be sketched on a cocktail napkin[i].

One of the most important processes in the design of an aircraft is its sizing. When the idea of a notional aircraft is conceived, it brings with it a set of requirements, even if they are relatively vague. An aircraft of any arbitrary size will not be able to meet all of these requirements. If the aircraft is too small, for example, it may not be able to fit all of its equipment or carry enough fuel for the distance it is supposed to fly. If it is too large, it may be impractical or too expensive. Making sure the aircraft is the proper size, then, is a critical exercise.

Volumetric concerns become important at this point. While there are certainly volume issues to be found in the initial phases of the conceptual design, it is only when the basic synthesis is complete and the aircraft has a basic shape that a true volumetric assessment can be performed. Many current design processes deal with volume either as an implicit calculation based on other

[i] This phrase is in deference to the number of designs (aircraft and otherwise) that have reputably been initiated by sketches on cocktail napkins.

relations or as something considered as a secondary, separate calculation. As will be shown later, there can be a more effective way to integrate volume into the design process that can be beneficial to a range of aircraft.

1.3 An Introduction to Volumetric Sizing

The concept of volumetric sizing at its most fundamental is simply the integration of volumetric concerns early in the design process of the aircraft. This means that the aircraft's capacity to carry, not only its designed payload, but also all of its necessary internal components, is considered important from the earliest phases of the design process. While traditional sizing has always addressed the weight of these components in some way, volumetric sizing adds the space they require to the calculations. Volumetric sizing can also contribute to the development of scaling laws that provide a simple, rapid way to estimate the size of an aircraft.

A volumetric sizing approach prevents designs that cannot carry their required load as well as those with "unexpected component placement" that require extra fairings, protuberances, or even external pods or other devices to accommodate components that do not fit in their anticipated locations. Understanding the volumetric qualities of a design also helps prevent significant (and expensive) changes to the aircraft's basic structure late in the design process. It also allows the creation of designs that are volumetrically robust – less sensitive to changes in the size or shape of internal components.

For traditionally constructed aircraft filling familiar roles, the idea of volumetric sizing is unusual. Most of these aircraft are extremely well defined from the earliest phases of design, and the empirical relations that have served for so many years will still apply. In fact, volumetric information has been integrated into the relations themselves, resulting in an implicit consideration of volume. With commercial jet transports, for example, volumetric concerns are of secondary importance at best, outside of the passenger compartment. Since most jet transports represent only evolutionary developments over prior models, much of the internal layout work has already been done, so volumetric concerns are represented in the traditional sizing methods with a relatively high level of fidelity.

For revolutionary designs or systems, familiar empirical relations may no longer apply. A revolutionary propulsion system may not scale in the same manner as the existing gas turbines and reciprocating engines do, and secondary systems, such as those for power conditioning on electrically powered aircraft, are also large components with little historical sizing data. Some of these systems are sized based on ground-based systems – a solution that involves a good deal of uncertainty[20]. These types of disruptive technologies can have a significant impact on the internal layout of even a large aircraft and can drive the selection of internal components[21].

Unusual missions could also benefit from a more comprehensive sizing tool – Charles Lindbergh made his transatlantic crossing in an airplane with no window in the front, but only a small periscope and the windows in the doors. His seat was directly behind a fuel tank. The poor visibility (even by the standards of the day) made landing extremely challenging. The design decision

was made in part to ease the center of gravity shift as fuel was burned during flight, but also to prevent Lindbergh from being sandwiched between fuel tanks and the engine in the event of a crash[22]. A more robust volumetric strategy, certainly not available in 1927, may have allowed him to fly an aircraft with better visibility while maintaining the safety factor he desired.

The "Guppies" and "Belugas" used by commercial aircraft manufacturers and launch vehicle operators to carry partially completed aircraft, fuel tanks, booster rockets, and other outsized cargo are examples of more modern volumetrically-driven designs. They have an odd, misshapen look to them as they were initially designed as heavy modifications to existing airframes.

Modern cargo aircraft are designed for volume constraints, often the need to carry a particular type of ground vehicle. The C-17 Globemaster III cargo aircraft had its cargo compartment height set by the "Apache helicopter with the rotor hub installed" while its load floor was designed to hold a M1 Abrams tank[23]. In both of these cases, the decision to carry a particular payload then influenced the remainder of the aircraft design in terms of both weight and volume considerations. The result, however, is the design of a space around which to wrap an aircraft, rather than the explicit incorporation of volumetric concerns for all of the aircraft's components.

An older SGT "Super Guppy" aircraft used to carry outsized cargo is shown in Figure 7. In the case of the original Guppy, an existing design was highly modified at significant expense in order to accommodate a particular cargo. The Super Guppy was a somewhat original design that used a large number of parts from other aircraft[24]. Development of this type of aircraft has continued, with the first flight of the Boeing Large Cargo Freighter, a heavily

modified 747-400 designed to carry parts of the 787 Dreamliner, occurring in September of 2006[25].

Figure 7: *A SGT "Super-Guppy" volumetrically-designed aircraft*[26]

Accounting for the inherent uncertainty in new designs and technologies is the primary motivation for volumetric sizing. The conceptual design phase involves the highest levels of uncertainty in the design process, as little information is available about the system so early in the process. Increasing the amount of volumetric knowledge available to designers early in the process will improve the overall fidelity of the design. Designing with volumetric concerns in

mind will also contribute to an increase in the design's robustness, allowing it to be less sensitive to changes in the size or shape of internal components later in the design process. The results of this work should enhance the designers' ability to accurately predict as early as possible the specifications and capabilities of a new aircraft. Such accurate predictions will ensure the best possible match for the aircraft and its desired capabilities.

1.3.1 Volumetric Sizing as Part of the Overall Design Process

Volumetric sizing as presented here is not a complete technique for describing the actual size of an aircraft. Nor is it uniquely connected to the notion of cargo capacity. Instead volumetric sizing is a component within a broader sizing scheme that helps overcome some of the limitations inherent in existing aircraft design methods. Rather than a complete replacement for parts of the sizing process that work, volumetric sizing provides information to the designer earlier in the design process – ensuring better results from that process. The basic placement of volumetric sizing in the overall design process is illustrated effectively by Nam in Figure 8.

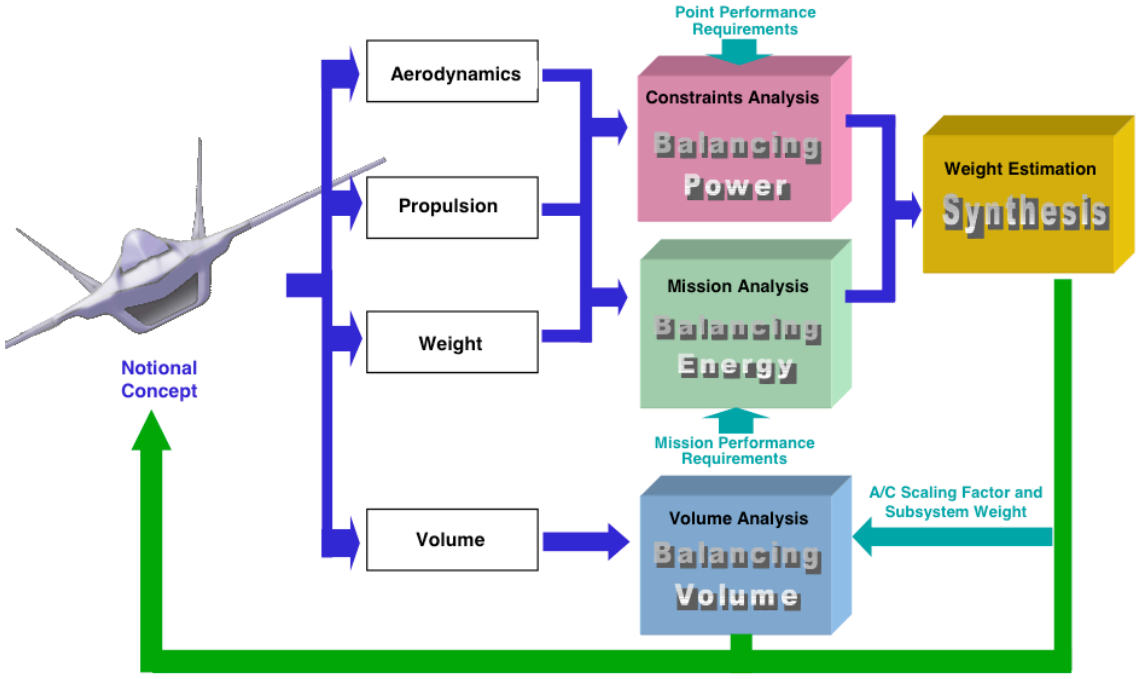


Figure 8: *Volume incorporated in the sizing process (after Nam)*[27]

The most important goal of volumetric sizing is to ensure that an aircraft design has enough space available for the internal components required to complete its chosen mission. This goal is inextricably linked to internal layout; the location and placement of internal components is the foundation for this work. Internal layout is currently performed at the end of the preliminary design phase and throughout the detail design phase of an aircraft's life cycle. This could mean that internal layout is performed months or years after the initial design is approved. Typically performed by hand (often with the aid of sophisticated computer-aided design software), and concurrently with the design of many of the internal components, a complex aircraft can take years to be fully described.

If an aircraft does not have sufficient space to complete a particular mission, only a few choices exist: the aircraft can be taken as-is, without the desired capabilities; the aircraft can be redesigned until it meets all of the stated requirements; or a series of trade-offs can be performed between redesigns and a relaxation of the mission constraints[28]. Typically the third choice is used, and aircraft can then be produced which cost more than the initial estimate, take longer to build, and yet still do not meet the original goals. If an aircraft has a surplus of space, the free space is normally considered wasted, as it is not put to use if it is not recognized until the end of the design process. There is some precedence for designing with "room to grow", and that becomes a balance between wasted capacity in the first units and growth potential in the later units. The F-15 aircraft was initially designed in this manner[29]. Designing with a goal in mind for the surplus space reduces the design's sensitivity to uncertainty in component sizing and can produce a more robust design.

One of the desired goals for volumetric sizing is an improvement in the ability to make trades against varying requirements. If, for example, a design has insufficient space for all of its internal components, volumetric sizing integrated with the rest of the sizing process would allow the rapid assessment of effects of needed changes on the overall performance of the aircraft. At the same time, a design with surplus space could be analyzed to determine if the space is better used for additional cargo or fuel, or if the space should be eliminated to reduce drag and improve cruise speed or some other parameter. In this way, surplus space is not wasted, but rather utilized to expand the aircraft's capabilities.

1.3.2 The Changing Design Paradigm and Volumetric Sizing

Design of nearly any original complex system can be described graphically as shown in Figure 9. As progress through the design process occurs from the initial requirements definition through manufacturing, more information is learned about the final product as decisions about its design are made. At the same time, design freedom is reduced as fewer alternatives are available with each successive design choice. Simultaneously, project costs are committed early in the process but not actually spent until the design nears production. Eventually, the system is fully described, costs are completely committed, and the design is in production[30].

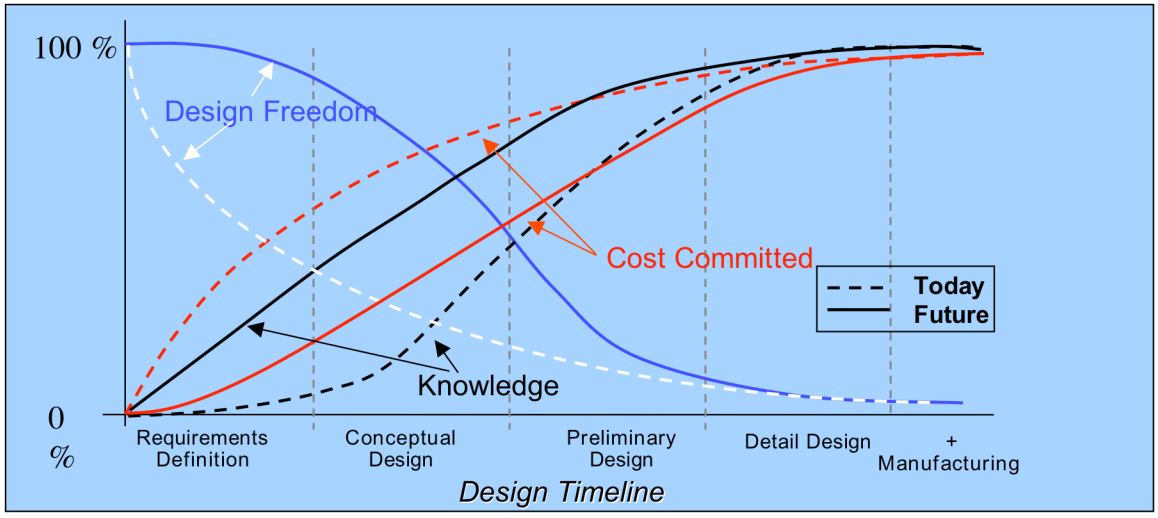


Figure 9: *The Shifting Design Paradigm*[31]

The so-called paradigm shift, illustrated above in Figure 9, is an attempt to improve the design process by shifting the lines of design freedom, cost

committed, and design knowledge in such a way that the overall design is improved without an increase in cost. The basic tenet of this idea is to increase knowledge about the design in the earliest phases of the design cycle[32]. This will allow the choices made about the design to be better informed, resulting in fewer mistakes and a better end result. Simultaneously, design freedom should be kept as long as possible in order to allow a wider range of options from which to choose the final design specifications. Keeping design freedom available as long as possible also delays the commitment of resources, delaying the time when the design must be frozen for financial reasons. Balancing these apparently contradictory desires requires a more complete understanding of the design earlier in the process[33].

One approach to coping with the difficulties embodied in the paradigm shift is the use of derivative designs. By using designs that possess significant commonality with others, designers can increase the amount of knowledge about a design early in the process. This results in the designers knowing a great deal about a design before work on it truly begins. Unfortunately, derivative designs also reduce design freedom in the early phases of the process. This is not always a desirable result, as early reductions in design freedom limit possible outcomes to those bounded by small perturbations from the original design.

Bringing knowledge forward in the design process allows better choices to be made regarding nearly every aspect of the design. A more fundamental understanding of the system being investigated prevents overly conservative decisions from being made simply because they have always worked in the past. Existing techniques cannot account for differences in designs that push past the limits of existing scaling laws and historical regressions. At the same time,

having a solid fundamental understanding of the design reduces the chance of basic errors being made due to a reliance on estimations, assumptions or low-fidelity analyses that do not apply to the design in question. Having knowledge about a design as early as possible allows the reduction of risk associated with a design while potentially improving the overall performance. Possessing this early knowledge also allows rapid response to changes in the design requirements, producing a final design that can be made more robust to changes in customer need.

Volumetric sizing fits well into this paradigm. In traditional sizing methodologies, volume is treated as a secondary or tertiary concern – something that can be worked out later in detail or that is handled implicitly in the early phases of design. The only components that are arranged in the earliest phases of design are the basic structure, propulsion systems, landing gear, and cargo/crew, and these initial configurations are often made for other reasons. Landing gear, for example, are placed in order to properly balance the aircraft about its center of gravity; actual detailed fit of the gear into their proper place comes later. Individual components, such as avionics, fuel tanks, and subsystems are either left until later or treated with simple regressions and engineering estimates. The disadvantage to this approach is that erroneous results from weight-based calculations can produce designs that are not capable of performing their stated missions because of a lack of space available for fuel or other mission-critical components. This causes extensive redesign late in the design process, which is time-consuming, expensive, and can result in a system that still does not meet expectations.

Complex analyses of aerodynamics, structures, and propulsion, enabled by rapid development of computational resources and their attendant software, have already improved the analysis capabilities of aircraft designers. Implementing a similar volumetric evaluation methodology will help designers ensure an acceptable internal layout for completing the required missions. The rapid advances in computer-aided design and manufacturing have produced extensive libraries of parts that are rendered with extremely high detail. At the same time, a volumetric sizing tool created with these technologies cannot manipulate components that do not yet exist, so for some components only an estimation of component size and shape is available. A trade must then be made early in the conception of a volumetric design tool to determine the level of complexity it will use.

The result is a necessarily simplified model that creates advantages for the aircraft designer without requiring so much time and effort that the advantages are overwhelmed by the effort required to attain them. In this way volumetric sizing is also similar to advanced aerodynamic or structural analyses: there is always a trade between effort and results, and the key is finding the balance that provides the best results within the possibilities of the available resources.

1.4 Current Techniques for Consideration of Internal Layout

The areas of an aircraft that are considered in a volumetric study change depending on the nature of the aircraft in question and the types of internal components it is expected to carry. For the "conventional" twin-engine commercial jet transport, for example, the engine nacelles include the engine and accessories – and really nothing else. The nacelles serve no real purpose in the volumetric sizing of a conventional transport as they have no space for components that should be in the fuselage. On the other hand, engine intakes and nozzles become extremely relevant for fighters and some other densely packed designs, as these tend to intrude on space that could be otherwise filled with fuel or payload.

Internally mounted reciprocating engines and electric motors tend to have a significant number of auxiliary accessories and cooling ducts attached that also need to be considered. One proposed aircraft design analyzed by the author, using a superconducting electric motor as its source of propulsion, required more volume for the motor cooling system than for the motor itself[34]. This suggests that whatever notional sizing tool is created to demonstrate the validity of the overall methodology should include the ability to pick and choose included components and volumes while ignoring those that have little impact.

Revolutionary aircraft designs also produce their own problems, as packaging equipment within unusual configurations is not a well-defined skill. Revolutionary subsystems also may not conform to common expectations with respect to size, weight, or secondary constraints. This increases the amount of uncertainty associated with the design when compared to aircraft with more

conventional components. A fuel cell propulsion system, for example, may be far larger and heavier than an equivalent gas turbine, and the accessories required to turn its power into thrust add other components that must find a home on the airframe.

In the simplest form, that of a commercial jet transport with under-wing engines, the only relevant spaces for a conceptual design study would be the fuselage and wing cavities (and possibly tail cavities). On the other hand, a blended wing-body design has a single exterior shape that can potentially contain a number of smaller, irregularly shaped compartments. Engines in this case would have nacelles integrated into the body, and secondary components that would normally be out on the wing may now share space with passengers or cargo in the main body of the fuselage. This can be seen to some extent in the F-22/Vega cutaways presented in Figure 2, above. The rear half of the F-22 is dominated by the engines and their movable nozzles. In contrast, the Lockheed Vega's rear fuselage is nearly empty.

Once the areas of interest in a notional design are chosen, a sizing tool must be able to determine which components will fit in the designated spaces. User-selected components need to be rearranged in whatever possible ways will allow them to fit within the necessary constraints. All of the landing gear, for example, could not be put in the nose of an aircraft as it would tip over, nor could they be placed on top of the aircraft without obvious undesired consequences. Arranging components becomes difficult as few items, apart from conventional fuels and other unpressurized liquids, can be shaped arbitrarily. The internal structure of each relevant space can also be an issue, with

protruding stringers, reinforcements, wiring and plumbing, or control linkages impinging on the available space.

One approach to this problem is to have a well-defined aircraft very early in the design process. Initially this sounds extremely practical, as advances in computer-aided design make the inclusion of digital data a relatively straightforward task. Unfortunately, this is rarely feasible, as the level of detailed data needed to define structural elements is usually tightly related to the placement of the very components being studied. A landing gear design, for example, is as dependent on where the landing gear is mounted relative to other components as it is to the weight it will be expected to support. The Boeing 767-400, shown in Figure 10 and B-52H, below in Figure 11, are similarly-sized aircraft, with the B-52 having a takeoff weight only about eight percent higher than the 767, yet their landing gear configurations are significantly different, as are their roles[35,36].

Figure 10: *Boeing 767-400*[37]

The 767 is a civilian airliner. It is designed primarily to carry passengers, who expect a relatively comfortable, pressurized space to travel. Additional cargo may come in the form of standardized cargo containers stored below the passengers. Placing the landing gear out of the way of the passenger and cargo space frees up room for more of this revenue-generating payload. Leaving the landing gear out of the way of any pressurized spaces also simplifies the structural design of the aircraft's pressure vessel. The B-52, on the other hand, was designed to carry a heavy load of bombs and a small crew. Bombs are relatively dense and take up little volume compared to civilian cargoes. They also require little in terms of environmental control. Placing the gear along the centerline of the aircraft results in a simpler structural design and less weight.

Figure 11: *NASA's Boeing B-52H*[38]

Defining a conceptual design where all of the individual components are completely determined is undesirable for the early phases of the design process. In addition to being extremely time-consuming, this option would remove a great deal of the inherent flexibility from the conceptual design process, possibly resulting in a sub-optimal design. With this concern noted, any proposed volumetric sizing methodology should have the capability to include standardized, predetermined parts if needed for cost, manufacturing or standardization purposes. It should also be capable of generating output useful in the later preliminary and detail design phases to help with precise placement and fit. In other words, any proposed volumetric sizing methodology must possess some measure of intelligence with respect to the use of volumetric data throughout the process.

A very basic form of intelligence can be achieved through use of one of several standardized file formats common among various CAD programs throughout the volumetric sizing process. The capability to pass internal layout information to later phases of the design will help ensure that knowledge created in the early phases of design is not wasted. Another approach would be to use the fundamental engine of a CAD program to manipulate components in a conceptual volumetric sizing exercise and then directly export the results to the CAD operators tasked with the preliminary design work. When used with the full capabilities of modern parametric CAD systems, this technique would become even more powerful, as structural and other constraints could be passed forward to a conceptual volumetric sizing tool. Unfortunately, little importance

is given to internal layout in the early phases of most traditional designs, so few of these techniques are applied.

A common approach currently used to predict fit of internal components is to study historical data and generate empirical relations to predict how much space will be available inside each compartment of interest – a wing, for example, may have a certain percentage of space available in a large transport, with contiguous spaces determined historically as well. This is similar to the technique outlined by Raymer[39]. A landing gear would have a similar historical sizing curve (as well as historical placement data). The unusable space within the wing is then considered to be taken up by structure, wiring, hydraulic lines, and space considered too small to be used. The historically sized landing gear in this example would then be checked for fit against the regressed space available in the wing.

A simple initial modification to this technique would be to use a distribution about the predicted volume coefficient. The assumed percentage of available space would then be bracketed on either side by an additional percentage to account for uncertainty in the initial prediction. This would produce a probabilistic result for any component fits; the result being dependent on which predicted distribution was correct.

In the case of a large commercial transport, the probability of fit would probably be quite high, as these designs are not particularly dense and are well defined. For a new fighter or uninhabited aerial vehicle (UAV) design, the reverse would be true. A low probability of fit would reflect the great uncertainty and tight confines related to such configurations. It also might simply be the result of a single component with a large amount of uncertainty – resulting in a

suboptimal design because of a single component. While wrapping uncertainty around exiting volume coefficients would be an improvement over deterministic techniques, it still does not sufficiently address all of the concerns that volumetric constraints might generate. There needs to be a better way.

1.5 Volumetric Sizing and Layout Within Existing Design Methodologies

Volumetric sizing has always been computationally difficult, but other problems have caused it to be often overlooked in modern design. For conventional designs that do not stretch too far from existing systems, explicit volumetric sizing is not required – the system is already too well defined for a rapid volumetric assessment to have any significant benefit. Resources that might have been used for volumetric sizing are instead put to work on the time-consuming detailed internal layout. Revolutionary concepts bring with them a host of other issues that have been, frankly, more important. Describing the internal layout of a notional aircraft system is of little importance when neither the airfoil required to lift it nor the engine required to push it exist. Most traditional sizing tools cannot properly handle truly revolutionary designs in any case, as the routines that are designed to model the size of an airplane's tail, for example, will not work properly for an airplane that has no conventional tail. The implied relationships that are often built into sizing programs to handle volume will also fail; for many tools, volumetric concerns become simply another task that needs to be considered with higher fidelity at some undefined point later in the process.

It is only when a design is sufficiently new that "standard practices" and popular tools do not work, yet sufficiently well developed that concerns over aerodynamics, structures, and propulsion do not overwhelm the volumetric issues that volumetric sizing becomes practical and necessary. This occurs when the design process has advanced sufficiently that the aircraft under consideration can, in fact, be analyzed thoroughly. An aircraft falls into this category when high-fidelity analyses are being performed on a host of design options early in the conceptual design phase. Rapid advances in physics-based tools for evaluating aerodynamics and structures have made possible rapid analysis of novel aircraft concepts, and these novel concepts have also increased the importance of accurate volumetric assessment. The problem has evolved from "will it fly?" to "will it fly as fast as I want?" Now it has become "will it fly the intended mission with the proper payload and equipment without bankrupting my company?"

Typically, initial internal layout is completed at the end of the preliminary design phase, but benefits of volumetric sizing to the design can accrue as early in the process as volumetric issues are addressed. In some cases, rearrangement of internal layout has occurred far into the production phase of the aircraft life cycle, and changing mission objectives often spark additional changes throughout the operational life of the aircraft. These changes are often accompanied by a substantial increase in cost as changes in a design are far more expensive when they are undertaken after production has begun. An excellent example of the correlation between the timing of changes and their associated costs can be found in the automotive industry. As Schrage shows in Figure 12, the Japanese auto industry is able to make engineering changes earlier in the

process than their American counterparts, a process that reduces the number of changes made and the amount of money spent making the changes[40]. Implementing volumetric awareness early in the design process can allow design changes to be predicted earlier, resulting in reduction of risk associated with a project. It is exactly this type of advantage that the inclusion of volumetric concerns early in the design process can bring to aircraft.

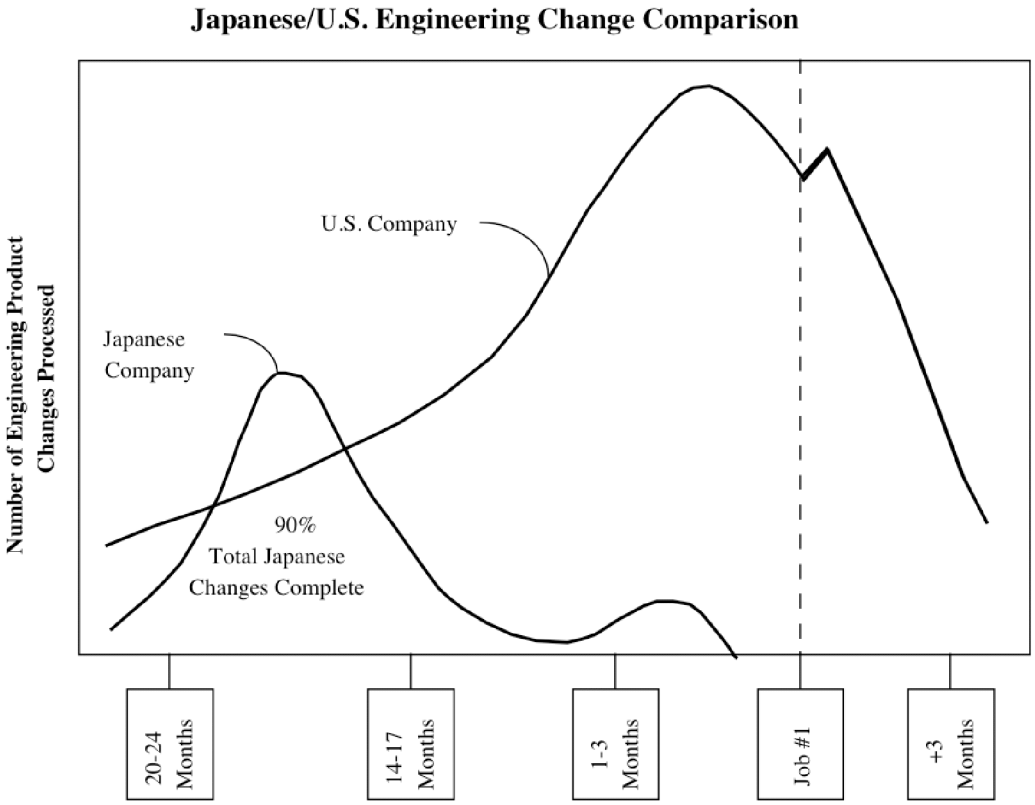


Figure 12: *Automotive Engineering Changes*[41]

Several existing tools are available for assisting with aircraft design, and there are methodologies to pursue as well that do not rely on a single program in order to analyze a particular design. Each of these techniques has its own characteristics, and while several have attempted to address volume in some way, none has really explored volume as thoroughly as it needs to be in order to fully integrate it in the design process. An overview of several design methods and techniques, as well as how they address volumetric issues, is presented below.

1.5.1 Roskam and Aircraft Design[42]

Jan Roskam's eight-volume series, *Airplane Design*, is widely considered the most extensive design work currently published, if a bit out of date. It presents the entire aircraft design process from initial concept to the end of preliminary design. With a base of empirical relations and historical trends gathered from its extensive collection of data on civilian and military aircraft from much of the history of aviation, Roskam's series covers most conventional airplane designs, and his work forms the basis for design classes at many universities. Designs that fall outside the range of the traditional used as the basis for the regressions may produce significant error when used with these approximations, however, as trends generated for traditional designs cannot accurately predict the performance of aircraft that do not share the same design assumptions. Completion of the series of tasks described in the eight volumes produces a well-developed design, nearly at the point of beginning production (No detailed study of full-scale production is included in any of the works

described here, though Roskam does address some of these issues in the eighth volume).

Roskam divides his work into two levels of fidelity, dividing the level of detail into "Class I" and "Class II". Work at "Class I" is with preliminary design tools, which "have limited accuracy but require only a small amount of engineering manhours". "Class II" work introduces methods that "have fairly good accuracy but require a significant expenditure of engineering manhours"[43]. Volumes 3-8 of Roskam's series are concerned with the "Class II" methods. The first two volumes cover the early conceptual phase of design and the "Class I" fidelity preliminary design. Figure 13 shows Roskam's design process in flowchart form, demonstrating the appropriate time for more detailed design work[ii]. Much of Roskam's work has been condensed into a software application currently available from his design company. In the Summer of 2006, Roskam's aircraft design and consulting company, DARcorporation, updated their *Advanced Aircraft Analysis* software to include rapid center of gravity calculations at the "Class II" level. While a great step forward in terms of speed from the calculation techniques presented in the printed books, the software still uses these calculations almost exclusively for weight- and control-related issues rather than volumetric fit concerns[44].

[ii] Somewhat confusingly, Roskam labels sections of the chart "Part I" and "Part II", which refers to the almost thematic division between initial sizing and more detailed, later work that is reflected in the division between the work in the first two volumes and that in the remaining six. Within the boxes, references to "PARTS" refer to individual volumes in the series.

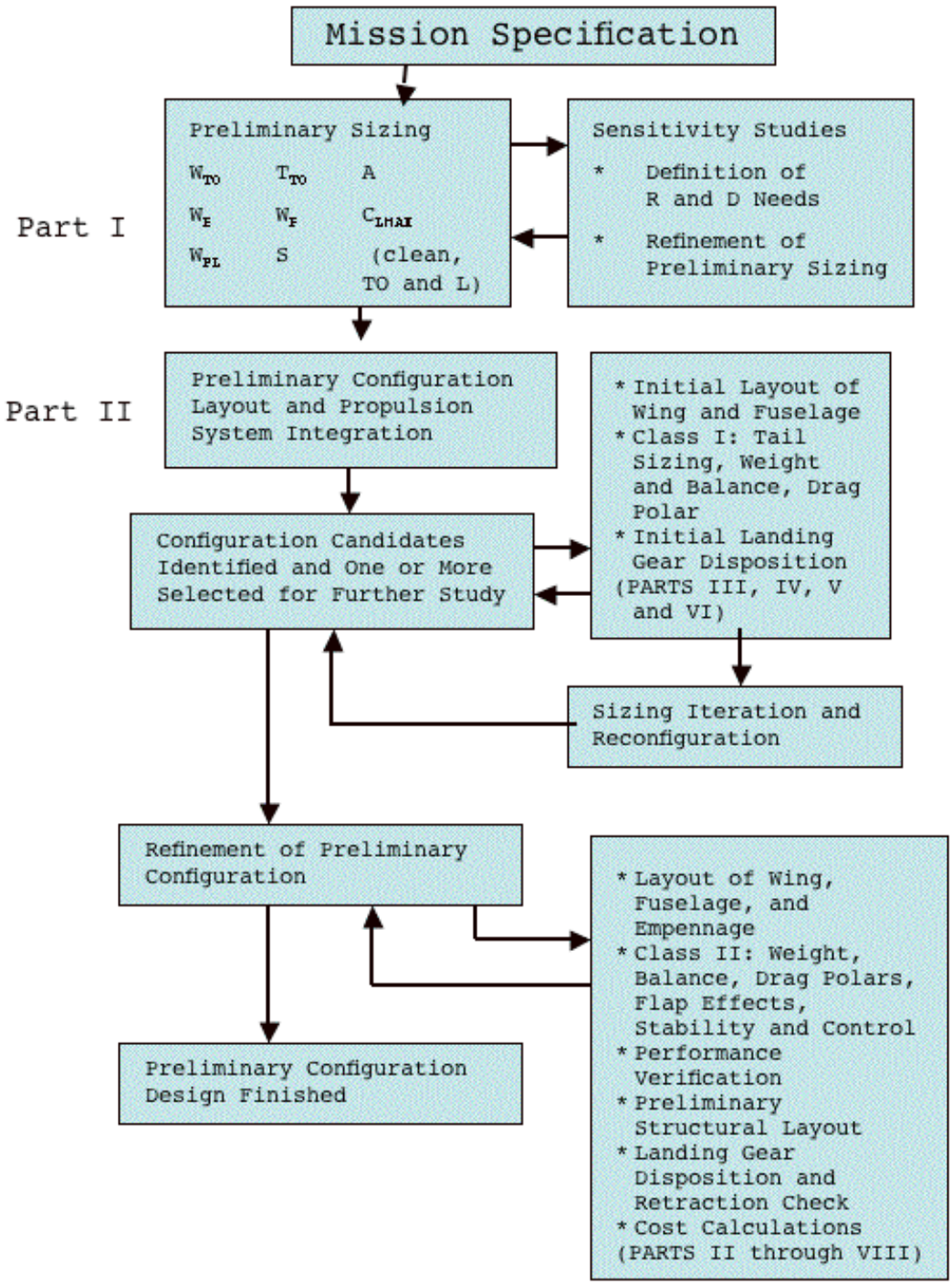


Figure 13: *Roskam's Preliminary Design Process*

1.5.2 Raymer and Net Design Volume

Daniel Raymer's single-volume approach, *Aircraft Design: A Conceptual Approach*, does not contain the vast number of charts, graphs, and drawings available in Roskam's *Airplane Design*. Instead, Raymer's text is more fluid, leaving the designer to find specifics such as passenger shape on his or her own. Raymer also includes significant stretches of exposition, resulting in a single volume that is extremely dense compared to Roskam's open, equation-laden work. Regressions are provided in Raymer's text, but they are typically in the form of only the equations, leaving out much of the raw data (some of which is actually taken from Roskam's series). Data that is provided tends to be tabulated or on smooth, manufactured charts. Raymer also does not include calculations to the depth of Roskam's Class II methods, yet they are more detailed than the Class I.

While his design text makes little mention of volume, Raymer touches on volumetric concerns in his thesis, *Enhancing Aircraft Conceptual Design using Multidisciplinary Optimization*[45]. His concept of Net Design Volume previews some of the concerns a volumetric sizing technique must address by including the volumetric impact of aircraft components such as engines, controls, and avionics. His suggestion is to collect volume data from every major component of the aircraft except the tail, assuming for the moment that the tail has almost no usable volume. This volume data would then be used for engine placement, controls, etc.

While this proposal has significant merit, it omits the most important components of an aircraft – its fuel, payload and passengers/crew. It also fails to address concerns of individual component placement, relying on empirical

relations for a single total volume of internal components. Using regressions through empirical volumetric data is a common thread within the design literature.

Internal components in Raymer's design text are sized using the same types of regression techniques found in Roskam's work, described above. One interesting tidbit, however, is a comparison between several regression techniques used to size a sample aircraft. Differences in the results of sizing regressions range from less than one percent to more than 50[46]. Obviously, some regressions do a better job of fitting a particular aircraft type than others. An advantage of a true volumetric sizing routine is that its effectiveness should be independent of the aircraft's type.

Raymer also comments on the efficacy of computer-aided design (CAD) systems for early phases of design, warning designers to "be wary of automatic CAD systems and always check the results for reasonableness using rough approximations..."[47] This is sound advice, and the importance of "sanity checks" helps guide the work here as well.

1.5.3 Volumetric Concerns in Ship Design with ASSET[48]

The US Navy makes use of their Advanced Surface Ship Evaluation Tool (ASSET)[iii] as a component in the conceptual and preliminary design of naval vessels. While it offers sophisticated 3D modeling of ship hulls for fluid and performance analyses, the internal layout capabilities of the program are much

[iii] The name of the software has recently been changed to *Advanced Ship and Submarine Evaluation Tool* to reflect enhanced capabilities related to submarine design. All references in this work, however, refer to the older version of the software.

more limited. While volume is one of the most important criteria for ocean-going ships, internal volumetric arrangement is less so. Taking advantage of the basic structure of most naval vessels, ASSET breaks the layout problem into decks, placing components in two dimensions on stratified decks, then integrating the decks into a complete ship. Aircraft designs are rarely stratified in this manner, so a simplified two-dimensional approach is ill-suited to aircraft design.

1.5.4 Howe's Aircraft Loading and Structural Layout[49]

Denis Howe's recent work is oriented primarily toward meeting governmental requirements and ensuring that aircraft are structurally sound in all aspects of flight and ground operations. While component mass and distribution are certainly critical parts of that, the focus is on structural analysis rather than on component layout. Howe suggests empirical relations for initial component layout studies, but admits that, "it is necessary to consider the introduction of new requirements in those cases where the concept of the aircraft is in any way unconventional"[50]. An important distinction inspired by Howe's work, that the component layout problems discussed here are of a non-structural nature, will be explained further below.

1.5.5 Technosoft's AMRaven[51]

AMRaven is a front-end to Technosoft's AML programming language. This allows access to the capabilities of the language with a minimum of overhead and a specialized set of tools specific to aircraft design. In many ways AMRaven combines much of the functionality of a CAD program with basic aircraft analysis tools, though it is not a dedicated sizing program. It can be used

for internal layout as well as a number of other aspects of aircraft design and offers built in integration with a number of optimizers and other analytical tools. At this point it is not capable of performing a complete volumetric sizing exercise without significant manual intervention, though Technosoft has demonstrated basic volumetric manipulation with AMRaven and fuel tanks[52].

1.5.6 FLOPS and ACSYNT

NASA's Flight Optimization System (FLOPS) is a flexible piece of software designed to size and/or analyze aircraft mission performance according to a set of aircraft and mission parameters. FLOPS contains data on three types of aircraft: commercial jet transports, fighters, and general aviation piston aircraft. Analyses are based on regressions through the available data and basic equations for aerodynamics, weights, control surfaces, etc. User-defined input is flexible and allows users to model a range of diverse aircraft. Volumetric concerns are all implicit in the internal calculations[53].

ACSYNT was developed at NASA-Ames Research Center during the 1970s to study the effects of advanced technology on aircraft synthesis[54]. While it is often preferred to FLOPS for military vehicles, the basic functionality between the two programs is quite similar. ACSYNT does contain a more sophisticated graphical component than FLOPS, and provisions have been made for consideration of major internal components, but the focus remains on outer lines for aerodynamic studies rather than on a true volumetric analysis[55].

1.5.7 Mattingly, Energy Methods, and the "Master Equation"[56]

J.D. Mattingly's *Aircraft Engine Design* is primarily an engine text and reference, and design elements are included primarily for the purpose of enabling the integration of a propulsion system with an aircraft. Mattingly's energy method involves the simplification of an aircraft into what is essentially a point mass with associated drag polar and simplified propulsion system. The aircraft then becomes a small system of equations that can be easily analyzed. This requires that the user know a great deal about the aerodynamic and propulsion characteristics of the aircraft before performing the analysis. While this level of detail is difficult to acquire early in the design process, rough approximations can be used to good effect with this method, making it useful for some aspects of the sizing process.

Using the notion of energy conservation and flow, Mattingly's technique allows a rapid assessment of potential performance characteristics of a particular design with a given propulsion system. The "Master Equation" is an expression of this energy method in a single, flexible equation allowing rapid assessment of aircraft performance in nearly any situation[57]. It is shown in its general form in **Figure 14**. In this form, T_{SL} is engine thrust at sea level, W_{TO} is aircraft weight at takeoff, β is the fraction of takeoff weight at a particular flight condition, α is the thrust lapse of the engine due to altitude effects, D is aerodynamic drag, R is resistance from the ground, V is velocity, h is altitude, and g_0 is gravity at sea level. The equation is extremely useful for generating point performance constraints such as rate of climb or turn. It is not, however, useful for detailed sizing, nor does it address internal layout, leaving that to other tools. Internal layout is present here only as implicit effects in the lift and drag values.

$$\frac{T_{SL}}{W_{TO}} = \frac{\beta}{\alpha}\left\{\left(\frac{D+R}{\beta W_{TO}}\right) + \frac{1}{V}\frac{d}{dt}\left(h + \frac{V^2}{2g_o}\right)\right\}$$

Figure 14: *Mattingly's energy method "Master Equation"*

1.6 Preliminary Solutions to Problems of Volumetric Sizing

Instead of limiting volumetric sizing to the "mechanical" components of an aircraft, a volumetric sizing method should also include the cargo, fuel, and personnel. While it would seem that these components would be included in any design exercise, including them in a formal conceptual design process eliminates the possibility of considering a design that cannot actually complete the desired mission(s).

At the same time, a distinction needs to be made between the structural components of an aircraft and the internal components to be studied here. While layout of wing boxes, internal bracing, and fuselage keels, for example, could be automated with tools similar to those presented in this work, the extra complexity involved in the structural analysis of such items precludes them from a study of this scope. Integrating a volumetric sizing code with a full structural analysis is an admirable goal, but not one that is feasible with the resources available for this work. Similarly, using a volumetric sizing approach to change

the shape of the fuselage/wing of the aircraft in question is beyond the scope of this work.

A conceptual volumetric sizing method must be able to account for uncertainty in both the size and shape of potential internal components. This flexibility will allow it to create a robust solution that can create a valid design despite changes in component size or shape. A proposed methodology must be capable of operating rapidly, yet still allow sufficient control over setup and operation to allow a wide range of potential design spaces. In addition, a conceptual tool for volumetric sizing should be able to operate with a great deal of configuration flexibility – it should not matter to the tool if the aircraft being designed is conventional or revolutionary – or even an aircraft at all. An answer to this challenge is posed in the next section.

2 Problem Definition

The basic concept of volumetric sizing is easy to understand. It is simply the inclusion of detailed volume information in the vehicle sizing process – determining if there is enough room in the airplane for all of the necessary fuel, cargo, passengers, and other internal components. The details of volumetric sizing can quickly become overwhelming, as many of the existing tools described below can be applied in varying degrees to a host of design problems. Similarly, the mathematics involved in bin packing are significant, and a good deal of research potential seems to exist simply in the field of optimizing particular types of bin packing problems. Deciding on a reasonable scope for the problem then becomes more important, as volumetric sizing takes place at various levels throughout the design process, from the earliest phases of conceptual design, to specific component layout in detail design, to cargo loading during the operational phases of a system's lifecycle.

2.1 Applications of Volumetric Sizing to Aircraft

For many conventional aircraft, volumetric sizing is simple and can be nontechnical. Once the initial design study has been performed, sketches or CAD descriptions of "normal" components are made within the bounds of the design to ensure fit. In many cases, historical precedence has predicted the component locations. Since most modern transport aircraft are not particularly dense, this

solution is often acceptable. Designs are usually derivative, and many components come in standard sizes or are easily scaled. This premise is used by hobbyist aircraft designers as well as by sizing programs such as NASA's *FLOPS*[58]. The process can be time consuming, however, and it fails to account for truly revolutionary designs. These designs, using new power sources, propulsion technologies, or construction techniques, may not have the great deal of surplus space found in many existing aircraft.

Commercial transports are not typically volumetrically constrained when considering component placement, as they are nearly always derivative designs making use of common construction techniques as well as an enormous body of empirical knowledge and historical data useful in planning internal layout, though there are exceptions. In the case of the Airbus A380, for example, nearly all of the internal layout work was completed before much of the rest of the design work[59]. In this case, the configuration of the aircraft depended greatly on how the cargo and passengers are arranged within the fuselage, which is the A380's most unusual feature. Adding a second deck of seats forced volumetric concerns to the forefront for that portion of the design; the remainder of the A380 remained similar to existing Airbus aircraft. This combination of product similarity and unusual shape made early component layout possible and necessary[60]. The large amount of product commonality between the A380 and other Airbus products made high-fidelity internal layout possible, as a great deal of similar work had been done on the company's other aircraft. The additional passenger deck made the design sufficiently different for the existing empirical relations to no longer be valid. A similar approach works well with large cargo aircraft. While in some ways similar to commercial transport aircraft, the bulk of

their systems are wrapped around an outsized cargo compartment typically designed to a single design mission.

General aviation aircraft have historically possessed a great surplus of internal space. With the exception of newer composite designs and some pusher-propelled aircraft, the long, tapered fuselage needed for correct placement of the tail surfaces has been almost completely empty. This can be shown with an informal survey of general aviation aircraft at nearly any functioning airport or in the examples presented earlier in Figure 4 and Figure 5. The relatively modest need for avionics and complex secondary systems on these aircraft also contribute to their great volumes of empty space. The low levels of performance and bulky crew compartments of these aircraft have also reduced the importance of extensive shape optimization. More modern aircraft have increased the density of general aviation aircraft somewhat, as can be seen in comparisons of aircraft from Cirrus Design, for example, and some of the older Cessna or Raytheon (Beechcraft) designs. The gentle slope of the fuselage from the cockpit down to the empennage on older designs contrasts with the significant taper of the newer aircraft. Higher wing loadings on many new aircraft and different construction techniques have also changed how fuel is stored in aircraft wings, changing the amount of space available there. How much the amount of unused space will change in designs now being developed is yet to be seen.

Advances in computer-aided design have also allowed relatively rapid placement of many internal components. Again, this type of solution becomes of limited use when faced with new construction technologies or revolutionary aircraft designs. Empirical relations detailing the proper number of aluminum stringers in a fuselage, for example, fall apart when designing a composite

aircraft. There are also no historical points from using similar component parts in similar locations, such as landing gear or avionics. In the A380, only the subsystem similarity between it and other Airbus aircraft allowed its rapid development, and even that was not a trouble-free exercise.

Fighters and other supersonic aircraft have far tighter constraints. Because of their increased sensitivity to fuselage cross-section, and concerns over handling qualities and detectability, fighters are extremely constrained. Fuel is put in any available open space in order to extend range and operational capability as much as possible. Supersonic transports are similarly constrained by the large amounts of fuel they must consume during a typical mission. Layout for these aircraft is extremely difficult, with detailed component placement necessary in order to give a good assessment of a design's qualities. The final design for the Concorde, for example, was not truly complete until the third aircraft had been built[61]. The F-117 Stealth Fighter is a good example of a volumetrically dense aircraft with unusual challenges. As seen in Figure 15, the F-117 is unusually shaped to emphasize its stealth characteristics. Practically every available space, save that in the tail, is used for a component or fuel. For a design that is so different from aircraft flying at the time, where would a designer go to find empirical relations for volume? This demonstrates the need for inclusion of volumetric concerns as early as possible in the design process.

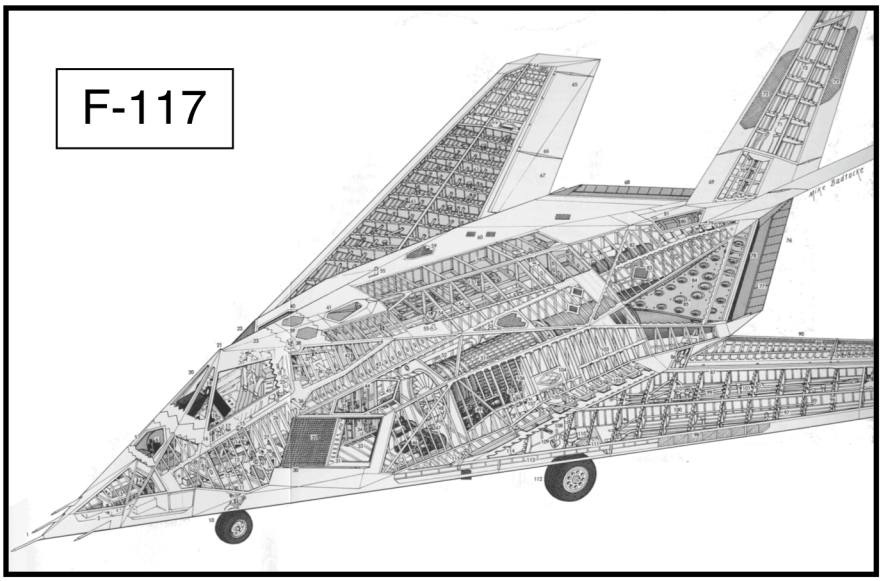


Figure 15: *Lockheed F-117 cutaway*[62]

Micro-sized uninhabited aerial vehicles (Micro – UAVs) have similar problems that come from a completely different need. Often using battery-powered propulsion systems, these aircraft have no "fuel burn" as such. Not only does this change the nature of component layout as tanks of fuel become batteries, it fundamentally challenges the traditional sizing techniques. No longer can a Breguet-style in-flight weight change be used to determine aircraft mission performance[iv,63]. With no measurable change in weight during a mission, battery-powered aircraft require a power-based sizing routine or some other alternate process. They also tend to be extremely densely packed, with batteries, sensors,

[iv] As explained by Anderson in *Aircraft Performance and Design*, the Breguet range equation is: $R = \frac{V_\infty}{c_t}\frac{L}{D}\ln\frac{W_0}{W_1}$, where R is range, W_1 and W_0 are the aircraft weights at the points of interest, V_∞ is velocity, c_t fuel consumption, and L/D is lift/drag. If the weight of the aircraft does not change during flight, the range becomes zero with this equation.

actuators, and transmitters stuck in every available space. The component count may be small, but errors in arranging the few components in a micro-UAV may cause significant changes in the aircraft's flight characteristics.

The same can be said for many lighter-than-air (LTA) vehicles, where the need for consistent weight throughout the mission (and the occasional use of water-trapping or other devices to maintain it) invalidates traditional weight-fraction range calculations[64]. Their large volumes also belie their significant volumetric requirements – the volume changes inherent in the use of lifting gases alone makes for a challenging problem in internal layout[65]. The empirical data for airship sizing is also extremely old, recommending development of techniques better suited to modern materials and design techniques[66].

2.2 The Relevance of Trucks and Commercial Shipping

A more approachable example can be found at any truck rental facility. U-Haul, for example, provides a chart, shown in Figure 16, that helps renters determine which truck they should use for moving their belongings[67]. Obviously, no rental company can know with any level of accuracy the specific items in a renter's household. Instead, truck rental companies rely on years of empirical data and a basic truck shape that has not changed in decades.

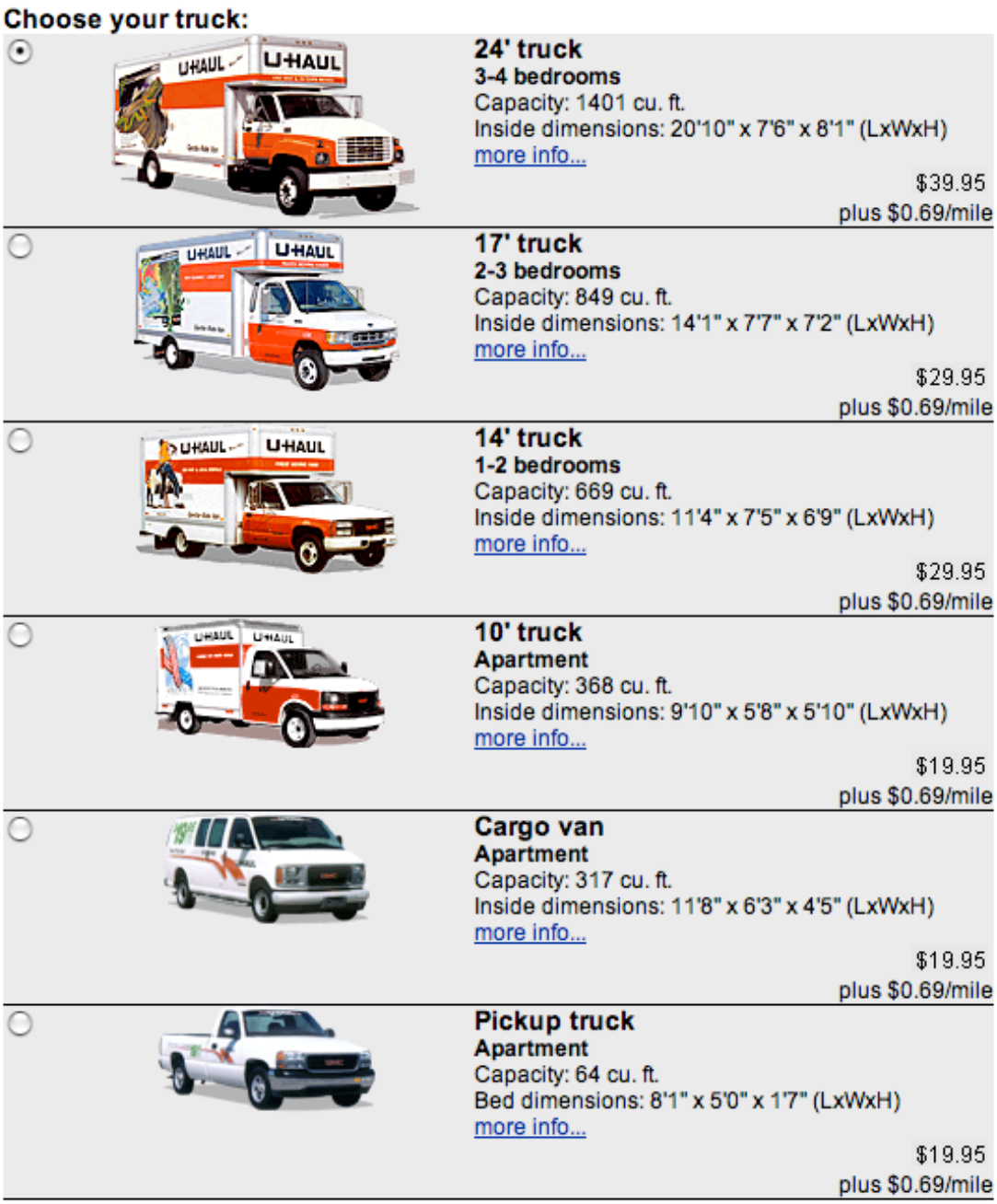


Figure 16: *Simple truck sizing at U-Haul*[68]

The situation is quite similar to conventional aircraft design in several ways: if a truck is too small, the renters will not be able to complete their move within the desired timeframe; if the truck is too large, the renters could have

saved money and effort by renting a smaller truck. Fortunately for both the renters and the trucking companies, this system works relatively well, as the consequences for renting a too-large truck are relatively minor and the empirical data is within its range of accuracy.

The simple truck model falls apart in the same place as the simple aircraft volumetric models – revolutionary concepts. If a renter has belongings that are extremely unusual in shape, size, or quantity, the empirical relations cannot accurately predict the correct truck size since they were not designed to account for the configuration needed by the renter. Similarly, the relations would not work for a truck design that eschewed the traditional box shape for something unusual, such as a cylindrical truck body or an inclusion (like a wing spar carry-through) in the middle of the payload area.

For renters with unusual needs (the trucks, happily, have remained rectangular and wingless), another method must be employed. Sizing for critical components is relatively straightforward: if a particularly long object is to be carried, a longer truck must be rented, and U-Haul does provide dimensions for this purpose[69]. Similarly, internal volume of each U-Haul truck is available for large quantities of materials. Here, simple back-of-the-envelope heuristics are employed almost naturally by experienced renters. More complicated approximations are also possible, including the use of software designed specifically for loading trucks.

In the same way, an aircraft might be sized based on a particular cargo or component item that drives the shape of the fuselage. The C-17 and Guppy mentioned in Chapter 1 are examples of cargo driving the size and shape of the aircraft. Likewise, the generous internal space available in either plane make

loading easier than in more volumetrically constrained designs. Designing around a particular cargo, while a form of volumetric sizing when applied as in the C17, is only a subset of the more general type pursued here.

Shipping companies using standardized shipping containers, cargo pallets, or truck trailers can choose from a variety of commercially available software packages designed specifically for the shipping industry. A Google search provides "about 218,000" results for "cargo loading software"[70]. Softtruck[71], MaxLoad[72], and LoadCaptain[73] are examples readily available online. Loadplanner.com even offers free web-based loading for education users who are not using it for commercial purposes[74]. One tactic used by most of the makers of commercially available shipping software is to assume that every item shipped will be in a box (or in bulk). This is a reasonable assumption, as most products sent by conventional freight methods are shipped in boxes, crates, or pallets. Some of these software packages are capable of adding constraints (such as axle loadings for trucks) and a few unusually shaped objects to their palette of capabilities, but none are well suited for aircraft design, as aircraft tend to be shaped differently from most commercial shipping containers (though the commercial software is often capable of loading standardized aircraft cargo containers). The commercial software surveyed is limited to known, monolithic spaces and relatively few constraints. Modeling a fuselage compartment and wing box while excluding a specified cargo area, for example, is not practical using commercial shipping software. The shape and orientation limits of most commercial applications also limit their applicability for volumetric sizing. Most shipping is done with boxes or pallets that are designed to lay flat on a flat surface, while aircraft have internal components of arbitrary shape, location, and

orientation. While this increases the speed of commercial shipping software, allowing rapid solutions on relatively modest computational hardware, it reduces the flexibility below that required for an aircraft tool. Volumetric sizing tools for aircraft will need a significantly different set of capabilities in order to be fully effective.

2.3 Volumetrics and Ship Design

Water-borne vessels have a much longer history of dealing with volumetric concerns than aircraft. Indeed, the ton, now a common measure of weight, was initially a measure of volume and used as the basis for taxing commercial shipping[v][75]. As most ships rely on the displacement of water in order to operate properly, the amount of volume enclosed by the hull becomes extremely important. While most cargo loading software is capable of loading standardized containers used aboard ships, most conceptual and preliminary ship design is still performed in a stacked two-dimensional arrangement. As mentioned earlier, the US Navy's *Advanced Surface Ship Evaluation Tool* (ASSET) uses this technique, saving a full three dimensional exploration to later in the design process[76].

[v] The English "ton" comes from "tun", a standard container for wine that occupied nearly 60 cubic feet and weighed 2,240 pounds.

2.4 Creating a Solution

Aircraft designers desiring to create revolutionary designs will need to consider internal volume in a more sophisticated way than that possessed by existing methods. The solutions that have served well for conventional aircraft and derivative designs simply cannot properly accommodate the increase in design fidelity required by new designs. The solution is to embrace volumetric concerns early in the design process.

Any attempt to address the task of volumetric sizing will need to accomplish several goals. It must be able to take a set of aircraft components and whatever space is available to fit them and find an acceptable arrangement if it exists. Unlike commercially available shipping and cargo-loading software, it must be able to take arbitrarily shaped objects and be able to place them in orientations that are not necessarily "flat against the wall". It also needs to take into account the types of constraints common to aviation applications. In addition, a true volumetric sizing solution must possess the following characteristics:

1) Rapid calculation. Solutions must be readily available, so the time to complete an assessment is minimized in the context of the time spent in the overall design process.

2) Flexibility. Whatever form the potential solution takes, it must be sufficiently intelligent to work for a wide range of vehicles. Ideally, the methodology would be applicable to any range of containers and their contents. Results should also be usable by other phases of the design process.

3) Accuracy. The process must work and be correct.

4) Uncertainty. The solution must be able to handle the uncertainty inherent in the descriptions and sizes of potential components in order to produce a robust solution.

These individual items, described in detail below, form the four basic practical goals for the end product of the work described here.

2.4.1 Rapid Calculation

Internal layout is a large, complex problem. As will be explained below, in section 4.2.2, internal layout is part of a class of problems called NP-Complete[77]. These problems cannot yet be solved with finesse; the only guaranteed solutions are brute force computations. The only other options are optimization techniques that cannot guarantee the best solution, only attempt to approximate it. At the same time, the long completion times characteristic of the brute force techniques are not useful in the context of the overall design process. For a volumetric sizing solution to be useful, it must not only work, but it must work quickly enough to allow its integration into the suite of other tools used by designers. Volumetric sizing in conceptual design is, after all, only a rapid approximation of the final layout, not a replacement for a complete CAD study.

This suggests the use of a domain-spanning optimization technique, which can operate over a large area and find the global optimum without being stuck at local optima. While these techniques typically cannot guarantee the best solution, they have shown excellent capabilities over a wide range of similar types of problems, being able to reliably find "good" solutions in far less time

than brute-force pattern-based or random techniques. A more detailed analysis of optimization techniques is presented in section 6.3.

A similar problem is found in the representation of individual component shapes. Fully describing a given component may be in itself a computationally significant task, so techniques are also needed to simplify the shapes of components being checked while minimizing loss of fidelity. The method as a whole needs to be generally rapid in order to demonstrate a performance advantage over expert-driven systems such as parametric computer-aided design.

2.4.2 Flexibility

While a tool that can operate on a specific vehicle has some use, the ability to generalize volumetric sizing techniques to a wide range of vehicle types adds flexibility and usefulness to the overarching methodology. This amount of flexibility also adds credence to the idea that volumetric sizing can be directly applied to revolutionary vehicles that have no record in the historical database. Results from a volumetric sizing exercise should also be usable by other components of the design process, allowing information to flow down to other tools from the volumetric assessment as well as up from other design tools. The intelligent processing of volumetric information is essential to a successful sizing methodology.

2.4.3 Accuracy

Producing correct results is important in any design process. With a task as unusual as volumetric sizing, few acceptable metrics exist for determination of

the technique's accuracy. The best (and most time-consuming) of these is construction of an actual prototype system. Numerical assessments in comparison to theoretical values are also important, as are analyses of experimental results. Each of these is expected to be used to varying degrees in order to verify the effectiveness of the basic ideas behind volumetric sizing. Any analysis tool developed in the pursuit of a volumetric sizing methodology must be capable of effectively and robustly producing accurate results when checking for intersections or any user-specified constraints.

2.4.4 Uncertainty

One element of the design process that is virtually guaranteed is that there will be a certain amount of uncertainty in each design. The uncertainty of relevance to volumetric studies comes primarily from a lack of knowledge about the internal components. How large, for example, will a hydraulic pump need to be in a particular design? If an aircraft is making use of a fuel cell power system, how large must the components be to properly manage the power conditioning needs of the propulsion system? What are the consequences of a fuel tank that is too small?

In this case, the only answers available may be those formed through regressions or estimations. In order to properly account for all the possibilities inherent in the combination of relatively coarse solutions, a volumetric sizing tool should be able to consider a range of component sizes and shapes when placing components in a space. This may reveal the importance of particular components in the closing of a design or help reveal a set of components crucial

in helping a design meet or exceed a particular goal. An effective method for volumetric sizing must be able to address at least some level of uncertainty.

The result of successfully addressing uncertainty in a volumetric sizing methodology is the production of a volumetrically robust design. While creating a design that is robust in the traditional sense is important – producing a design that is resistant to changes in mission or technology specification, producing a volumetrically robust design also increases the usefulness of the final product. A design that can be produced successfully with a wide range of values for size and shape of internal components can be considered volumetrically robust, and reduces the risk present due to the uncertainty inherent in any new design.

3 Hypothesis and Research Questions

Aircraft design is inherently a difficult and continually evolving task[78]. Traditionally volumetric sizing has been handled through implicit techniques and regressions, basing the component layout of the next design on the layout of the last one. In his sample analysis of a twin-engine light transport, Roskam disallows a joined-wing configuration "primarily because of lack of a data base"[79]. A need exists for a more sophisticated solution. Current methods do not sufficiently address this need, and existing methods in other fields lack the sophistication and flexibility needed for aircraft design. Thus, the following solution is proposed: *An intelligent, robust volumetric sizing methodology can reduce design uncertainty and improve the quality of the final design by bringing internal layout information forward in the design process.*

This hypothesis can be further explored with the following four lemmata:

1) *Components in a container can be approximated to allow a more rapid exploration of the design space.*
2) *These components can be assigned a degree of uncertainty to account for changes in size and shape with improved levels of knowledge.*
3) *A relatively robust and efficient system can be developed to check components for fit within the boundaries imposed by their system.*
4) *This system should require less time and effort to run than a traditional CAD layout of every component and should be acceptable for the conceptual phase of system design.*

This hypothesis and its associated lemmata are supported by a set of four basic research questions:

1) Can volume-based information be included early in the conceptual design phase of a system design?

2) Can this information be used to improve decision-making early in the design process?

3) Can this methodology be applied rapidly and easily using readily available computing resources?

4) Can the results then be passed on to other phases of the design process in such a way to improve the efficiency of the overall design process?

The problem addressed in this work suggests the development of a methodology to rapidly assess notional designs to ensure sufficient space available for all the required internal components. The volumetric sizing tool to implement this methodology must contain the four characteristics mentioned in the last section: *Rapid calculation*, *Flexibility*, *Accuracy*, and the ability to address *Uncertainty*. It should also be structured such that its design and construction help answer the above research questions. While the prototype tool created to demonstrate this methodology should conform to the above requirements, this work is not designed as the development environment for a commercial software application but as a demonstration of the validity of the methodology on which it is based. It is also not designed to replace a CAD analysis, but rather to complement it, using elements of a CAD environment for a rapid assessment and then passing the information back to the more detailed tools.

On a practical level, the techniques used in the development of all components for this work are designed to be as broadly applicable as possible. Software and hardware frameworks have been chosen with a view toward

generality whenever possible in order to facilitate the extension of these ideas to as many specific applications as is practical. In a similar vein, software tools that require expensive or restrictive licensing or have severe platform restrictions have been avoided whenever possible to reduce potential issues when running examples in a wide variety of locations and on a variety of hardware platforms. The hypothesis and research questions are summarized in Table 1.

Table 1: *Hypothesis, Lemmata and research questions*

Hypothesis and Lemmata
An intelligent, robust volumetric sizing methodology can reduce design uncertainty and improve the quality of the final design by bringing internal layout information forward in the design process.
1) Components in a container can be approximated to allow a more rapid exploration of the design space.
2) These components can be assigned a degree of uncertainty to account for changes in size and shape with improved levels of knowledge.
3) A relatively robust and efficient system can be developed to check components for fit within the boundaries imposed by their system.
4) This system will require less time and effort to run than a traditional CAD layout of every component and should be acceptable for the conceptual phase of system design.
Research Questions
1) Can volume-based information be included early in the conceptual design phase of a system design?
2) Can this information be used to improve decision-making early in the design process?
3) Can this methodology be applied rapidly and easily using readily available computing resources?
4) Can the results then be passed on to other phases of the design process in such a way to improve the efficiency of the overall design process?

4 Problem Decomposition and Methodology

For the purposes of this work, the proposed volumetric sizing methodology can be applied as a supplemental technique to those employed by traditional aircraft designers. The basic outline for this process is shown in Figure 17. The traditional design process provides a set of internal components for sizing and a geometric container for placement of the components. The volumetric sizing tool then utilizes automated layout techniques and a local optimizer to obtain the most successful internal arrangement available for the aircraft. The results of this optimization may have consequences on the performance of the aircraft through alterations in aerodynamic characteristics, center of gravity and moments of inertia, or even placement of the wing or landing gear. Relevant results and component locations are then returned to the traditional design tool for further analysis and selection of a candidate design. Eventually, volumetric sizing should be transparently integrated with traditional design techniques.

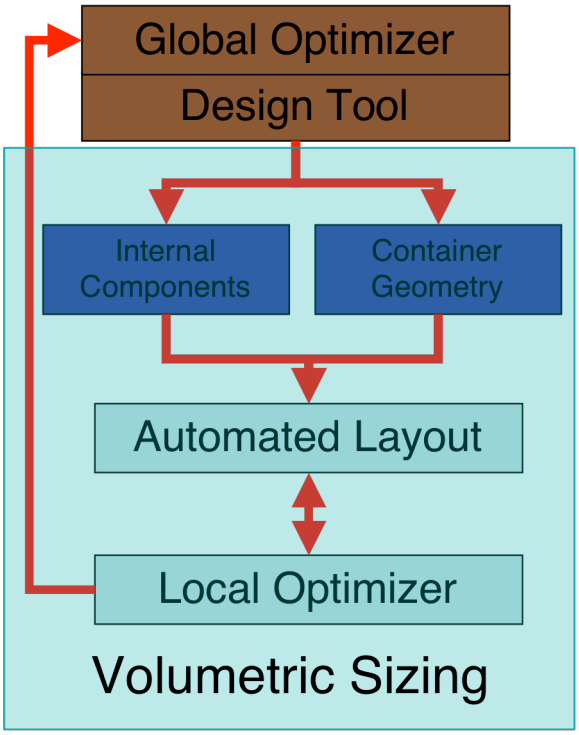


Figure 17: *Volumetric sizing as an addition to the design process*

In any volumetric sizing exercise, there will be a container and a set of components to be placed. In the methodology described above, these are produced by the traditional design environment, although in a fully integrated design tool, volumetric concerns would be considered throughout the design process. Evaluating a particular container for suitability with its associated components, whether the container be a simple box, a fuselage, or an entire aircraft interior, can be rendered into the same sets of actions, regardless of the tool used to implement them:

1) *Definition of container*. The space that will hold the components must be defined. In the case of an aircraft, the space will probably be either a fuselage or a fuselage/wing combination.

This needs to be possible at nearly any level of geometric complexity, to account for internal structural supports, for example. It will remain relatively simple for the purposes of this work in order to facilitate a rapid solution.

2) *Definition of internal components*. The components that will fill the container must be defined at arbitrary levels of detail, from rough geometric solids to complete parts. These items need to be constructed to be able to take into account uncertainty in both size and shape. In a fully integrated volumetric design process, internal components may be brought forward from an existing CAD library.

3) *Placement of components*. The internal components have six degrees of freedom within a three-dimensional space – three for location, and three for orientation.

4) *Verification of components*. Once placed, the components must be checked for intersection with their neighbors and the walls of the container.

5) *Rearrangement of components*. In the likely event that a particular arrangement is unsuitable, a method to find a better solution should be created with a technique for ranking unacceptable arrangements in the event no solution exists.

6) *Optimization*. The best solution should be presented. In the event that no arrangement of components works, this will lead to a revision of the container or a reevaluation of the mission goals. In the event of a successful arrangement, an accurate estimate of the extra space available should be produced.

7) *Constraints*. The method should be capable of dealing with customer-required constraints, such as those for placement, weight, thermal tolerances, or wiring.

These tasks combine to generate a solution to a particular internal layout problem. They are designed to operate within the scope of a broader design scheme and should also be arranged to integrate as seamlessly as possible with other tools founder later in the design process. Many such layout solutions can then be assessed to determine the best design for a particular set of internal components and this information can then be passed on to the preliminary design toolset.

4.1 Exploring the Combinatorial Space

Once the available spaces have been defined within the aircraft, and the volumetric uncertainty of individual components calculated, attention turns to the actual placement of components. Some components will be of fixed size, while others will scale with wing area, gross weight, or some other set of factors. Constraints on placement will also influence the overall arrangement of components within the space. The result is a highly complex space with an almost overwhelming combinatorial expansion. Even positioning a single box in the ten-foot U-Haul truck shown in Figure 16, assuming a tenth of an inch placement resolution and the limited rotational choices typical of rectangular boxes, would yield approximately $5.6*10^8$ possible locations. For components of arbitrary shape and possible arrangements, and for areas with more "spaces" than components, the dimensionality increases dramatically. Placing 10 arbitrary components in a space the same length as a Boeing 777 to an accuracy of a tenth of an inch in the longest direction, with rotations limited to no finer than one

degree, and with no constraints will result in about $1.16*10^{199}$ possible arrangements[80].

The initial impulse is to take advantage of exhaustive CAD part libraries and simply try to fit "stock" components into the new design, thereby eliminating at least component shape uncertainty from the work. While this is useful for preliminary design (and absolutely required for detail design), it is again computationally intense for the conceptual phases of design, where thousands of potential designs would need to be evaluated simultaneously. Most computational techniques for determining component intersection are highly dependent on the complexity of the individual components, so a fully-described CAD part may add significant computational overhead to the analysis. A flexible and simplified model of a part may be just as useful in initial fitting while requiring less computational time for fit analysis. In a fully-integrated example of volumetric sizing, a CAD library could provide the basis for the components used in the volumetric analysis, with the results feeding back to the CAD environment for later use.

A probabilistic approach to part sizing can ameliorate some concerns associated with using coarsely-defined parts. Each well-defined CAD part can be coarsely sized over a particular range by whatever variable drives its scaling (gross weight, wing area, aileron force required, etc.). Complex parts can be simplified through techniques common to the computer graphics industry and then rapidly scaled. Standardized parts that are relatively uncomplicated can be represented by simple deterministic models. By roughly defining a part with a parametric sizing distribution, studies can be rapidly performed to determine if redesign of a particular part is required, desired, or unnecessary. For volumetric

purposes, most of the internal characteristics of internal components are unnecessary, so a simpler surface representation can be used for most calculations.

An intelligent system is also needed to avoid increasing dimensionality beyond what is computationally realistic. Some design scenarios, for example, can be excluded on the basis of very simple calculations. Excluding these "easy" cases should reduce computational time considerably. Simply summing the volume of individual components and then comparing the sum to the total available volume creates a lower bound – no rearrangement of the individual components will result in a smaller volume, so aircraft volumes that will not hold this liquid volume equivalent (LVE) of the summed components will not hold the components themselves. If a set of distributions is present for the set of components, a simple study will need to be performed to see if further efficiency can be achieved by calculating the LVE for the largest combination of factors, or if the smallest portion of the distribution should be used.

Similarly, creating rectangular boxes around each component reduces the complexity of the calculations and offers the possibility of simplified alignment. These "boxed" components could then be arranged in a simplified, automated, three-dimensional, Tetris-style game or selected through a domain-spanning optimization algorithm. The Tetris-style solution is attractive intuitively, as it seems to match the problem conceptually. Indeed, Demaine, et al. have demonstrated that Tetris is an NP-hard problem and can be approached in a similar fashion to the problem of internal layout[81]. In three dimensions, however, a Tetris-style solution becomes more difficult to implement, and it carries the possibility of pushing all of the components to one end of the container when

that is really not what is desired – the solution technique itself in this case may interfere with the needs of physical constraints on the airframe.

A genetic algorithm or other domain-spanning optimization technique may be a better solution since the task is not necessarily to find the optimal design point but rather to simply find a "sufficient" design. With modern computational capabilities, this technique may be possible in the conceptual design phase, a key element of this work. If these boxed volume equivalent (BVE) components fit in the aircraft volume, the components themselves will certainly fit – even if arranged somewhat differently. This forms a limited upper bound on the required volume. Only if the BVE does not fit while the LVE does would more complex calculations be required. Uncertainty can be included in these measurements as well, providing a simpler technique for estimating required volume. This technique would work best in an unconstrained environment, where strict rules about individual component location would be less likely to interfere with the simplicity inherent in this approximation.

If finer resolution is desired without resorting to distributions of dimensional uncertainty, more finely discretized versions of the components, the simplified volume equivalents (SVE) would be generated and the system run again. These discretized forms can begin with basic geometric shapes (cones, cylinders, cubes, simple triangular meshes, etc.) representing the outer limits of components. These shapes can be easily manipulated, and their associated characteristics are more easily (and rapidly) calculated than those of the detailed CAD model. Expanding back out of the deterministic world, simple distributions can be used around the BVE values to estimate the actual component sizes without the need for complicated (and computationally expensive) CAD detail

work. This is also an improved way of dealing with the relatively simple shapes likely to be part of the conceptual design phase.

The BVE and subsequent detailed discretizations will become more computationally complex, yet dimensionally reduced, with the inclusion of position constraints. While requiring more intelligence from the system arranging the components within the simulated aircraft volume, the constraints remove combinatorial sets that add time to the analysis. By requiring a component to be within a certain position (or range of positions), the analysis will not have to consider combinations that would leave the constrained components in the wrong place. Reducing the number of directions of rotation will have a similar effect as parts are placed in a coarser manner. Discovering the optimum point on the curve between dimensionality and complexity while allowing sufficient arrangement resolution will determine the most efficient analysis. The desired analysis fidelity becomes an input to the process, with a working tool using simplified geometries of varying fidelity to verify sufficient volume in a design. Output will be a level of confidence for each design. An advanced tool will indicate which component is causing problems, if such a component exists for a particular design.

A similar approach can be taken with a cost model. If components do not fit within a particular container, or if they violate a constraint, the cost of the system due to redesign will increase. This can be combined with a production model that also reflects changes in production costs attributed to redesign. In this case, the effects of uncertainty would be seen as changes to costs within the system. These costs could either be represented as real dollars or as a utility function reflecting the performance impacts of design changes.

4.2 Application of a Volumetric Sizing Methodology

With the foundation in place showing where volumetric sizing can fit in the profile of an existing sizing process, and the basis created for approaching internal component layout, a more structured look at the fundamental methodology is required. The core of any volumetric sizing methodology begins with a set of geometric data for a container and a set of data for the components that will fill it. At the end, the results should provide modified volume information to the sizing tool and preliminary internal layout for use in later phases of the design process. The whole process should fit within a sizing methodology much like that shown in Figure 8.

The fundamental mechanism at the heart of volumetric sizing is a form of simplified automated internal layout. The internal layout process produces the results utilized by the rest of the design tool to explore the consequences of including volumetric effects in the initial phases of the design.

Figure 18 demonstrates the basic flow of automated layout within a more traditional sizing methodology; the steps are outlined in Table 2. Once the results from this portion of the analysis are compiled, the real power of volumetric sizing can be demonstrated. While performing a basic automated internal layout at the conceptual phase of the design process is useful, using the results of the internal layout exercise to affect later design choices demonstrates the effectiveness of volumetric sizing.

In the work presented here, volumetric sizing is handled as a separate element of the sizing process, and layout information is passed among design

tools by hand, shared file formats, and simple metadata associated with each component. In future implementations, volumetric sizing should be fully integrated, with supplemental data passing back and forth seamlessly between volumetric elements and later preliminary and detail design layout tools.

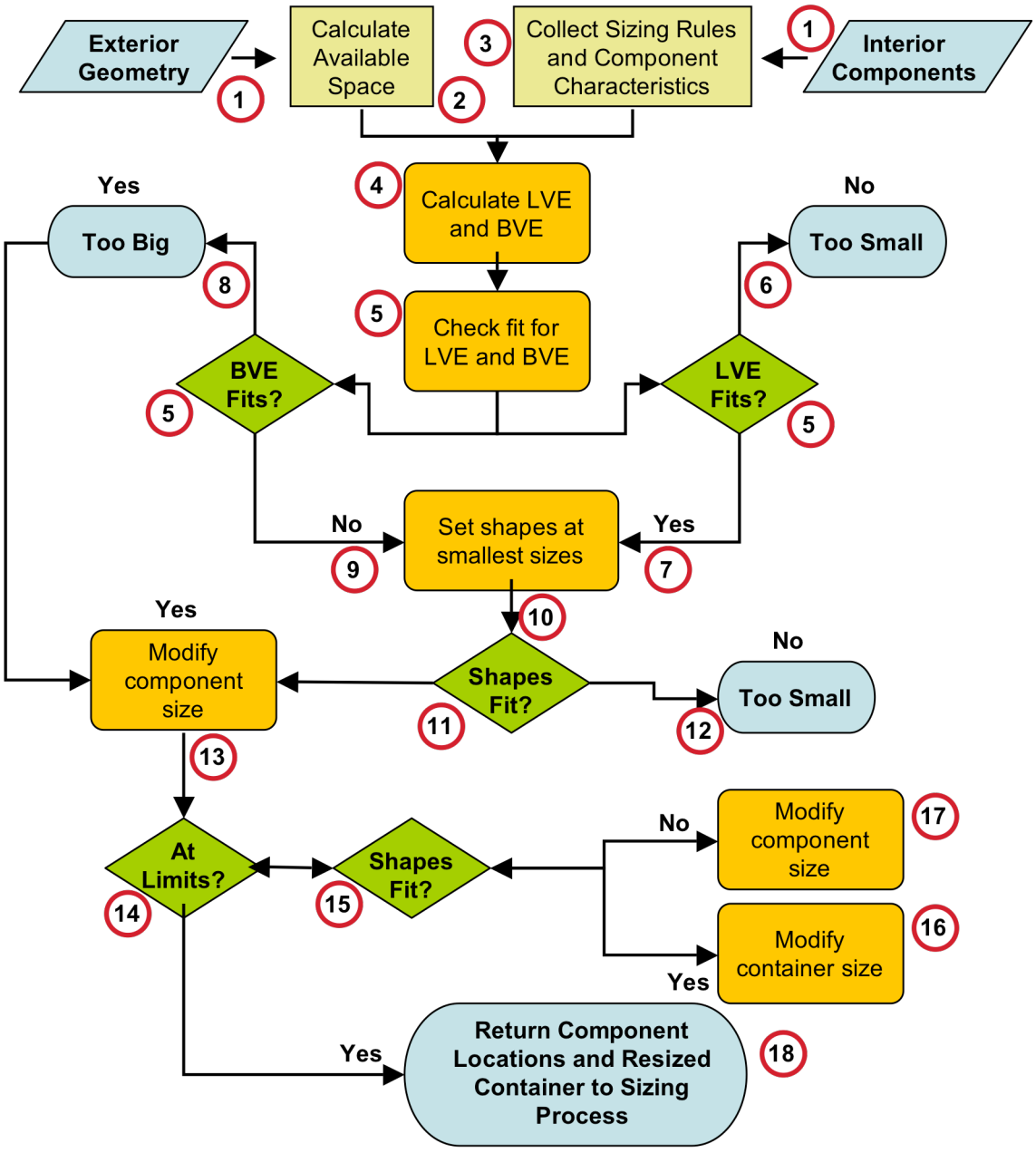


Figure 18: *Flowchart of volumetric elements in a sizing process*

Table 2: *Steps in automated internal layout*

1	Define exterior geometry and collect component data
2	Calculate available space and define interior container
3	Collect detailed component characteristics and size distributions; develop related constraints and collect component sizing rules.
4	Calculate liquid volume equivalent (LVE) and boxed volume equivalent (BVE) values for all components
5	Check fit for LVE and BVE
6	If LVE fails, design is much too small… Exit the loop and revisit the initial design parameters
7	If LVE passes, continue
8	If BVE passes, design is sufficiently large (but may be too big…). Modify the container space and continue.
9	If BVE fails, continue
10	Set the components at the smallest sizes as described by their size distributions
11	Check fit for shapes in the current configuration
12	If smallest components fail, container space is too small… Exit the loop and revisit initial design parameters
13	If smallest components pass, design is sufficiently large, increase component size where allowed
14	Check component size limits
15	If within limits, check new components for fit
16	If components fit, reduce container size, if desired
17	If component fits fail, reduce component sizes
18	If at component and container limits, exit the loop and return to the sizing process with component parameters and container modifications (if any)

4.3 The Mathematics of Volumetric Sizing and Internal Layout

The fundamental mechanism behind volumetric sizing, the internal arrangement of aircraft components, is a relatively well-understood problem in mathematics and computer science, though it is often expressed in the abstractions common to those fields. While the specifics that make aircraft internal layout a design challenge are not a part of the literature in these fields, both areas can make significant contributions to the progress of this work. The challenge is to develop an intelligent approach to the solution that improves the design process by taking advantage of the lessons learned from existing aircraft design techniques and from experiences found in computer science.

4.3.1 Internal Layout and the Knapsack Problem

Placement of internal components in a space is an extension of the bin-packing problem. A commonly studied problem in computer science, bin packing has applications across a wide variety of fields, though it is found most often in connection with industrial processes. It is commonly expressed in zero, one, two, or three dimensions, with increasing difficulty. In the zeroth dimension, bin packing simplifies to the "knapsack problem" [82]. A notional knapsack is provided with a fixed weight capacity. The task is then to fill the knapsack with a variety of objects of varying value in a way that makes use of as much of the sack's capacity as possible. In this case, order is not particularly important, as weight and total value are the only measure of importance, though

variations of the problem involve other constraints that complicate the solution[vi,83].

The *one-dimensional* variant involves lengths of string, wire, or other quasi-one-dimensional objects from which segments are cut to minimize the overall material used. Cutting lumber for use in building a deck, for example, would make use of one-dimensional bin-packing techniques. Order is important here, and a number of empirical relations exist that are commonly used to simplify calculations. These empirical relations may result in sub-optimal solutions, but they significantly reduce the calculation time, and the solutions are often close to the optimum, so the small penalty in solution efficiency is often worth the savings in calculation time.

In *two dimensions*, bin packing is commonly used to maximize the effective use of areas of material being segmented into individual components. Cutting cardboard sheets into packaging materials or fabric into sections of clothing are common uses. In these cases, arrangement of individual items is critical to minimizing the amount of material used. Ordering becomes more important as patterns become more complicated and empirical relationships become less effective. Heuristics, a set of loose rules assisting the process, become more important as a "good, fast" solution can become more important than "the best" solution.

[vi] The author's first exposure to this problem was as a child through a NASA version now called *Survivor on the Moon*, involving a notional crash landing on the Moon. Participants are asked to rank the importance of 15 components to carry in an attempt to reach the sanctuary of the "mother ship" 320km away. This problem differs from most knapsack problems in that the utility of the components selected is not revealed until the end of the exercise.

In *three dimensions*, the problem becomes even more complex than in two dimensions. Simple improvements to algorithms designed to find the best arrangement become even less efficient, and the number of possible solutions increases geometrically. Here, the problem is usually to find the smallest container (or number of containers) that can hold a group of components. In its simplest form, it is only the increase in size that makes the three-dimensional (3D) bin packing problem difficult – there is nothing conceptually challenging about adding the third dimension.

Unfortunately, the conceptual simplicity that applies to the basic problem does not apply well to the heuristics and other empirical relations that are common in the one- and two-dimensional problems[84]. One common solution technique for certain types of 3D bin packing is to simply form the basic three dimensional space into stacked layers of simpler, two dimensional problems. This is the solution used by Li and Antonsson in their exploration of genetic algorithms for microelectromechanical systems (MEMS) synthesis[85]. This solution can also be used in places where stacks of nearly two-dimensional objects occur naturally. Microchip design, where components are embedded in layers of silicon (much as in the MEMS example above), is well-suited to this approach, as is the design of oceangoing ships[86,87].

Many real problems involving 3D bin packing also add important constraints that make simplifying algorithms less effective. For example, an algorithm that chooses the largest item first to fit into a container may violate constraints dealing with mass distribution. These constraints can become so significant that they make fundamental changes to the nature of the problem, an issue that will be explored later.

The specific problem of internal layout, whether for aircraft or anything else, also changes the nature of the bins somewhat. While much of the work on bin packing solutions is oriented toward industrial problems involving large numbers of bins containing (usually) large numbers of components, internal layout usually involves one or only a few bins containing many individual components. The common bin packing technique of placing an inconveniently-shaped component in the next bin simply cannot apply when there is only one bin. Components in an internal layout problem are also more likely to be of widely varying sizes and shapes, while many traditional 3D bin packing problems involve large numbers of identical (or nearly so) components.

Internal layout problems can be initially designated as offline problems, meaning that component sizes and shapes are known ahead of time. This is a tremendous advantage in many bin packing scenarios, as it allows the components to be arranged in the most advantageous order for placement. The idea of attempting to calculate internal layout of an aircraft with no concept ahead of time of the shape of individual components seems nonsensical, but the distinction between online and offline problems for internal layout can get blurred in some cases. The usefulness of ordering, for example, gets called into doubt through the inclusion of constraints, and the basic notion of internal layout of an aircraft has fundamentally different goals from traditional bin packing, even though the mathematics involved in the solutions of both problems are similar. The specific issues found in problems of internal layout will be described below.

For offline problems in general, the work on online bin packing problems, such as that by Seiden[88] or Bentley, et al[89], which focuses on ratios of performance

vs. the best case, becomes less relevant in terms of their raw performance measures. The constraints and conceptual differences inherent in internal layout when compared to traditional bin packing simply make many of their algorithms inappropriate for this work. Portions of the algorithms they suggest, however, can still be applied to internal layout problems, and this will be explored in detail in a later section.

4.3.2 Bin Packing as a NP-Complete Problem

The general bin packing problem and by extension the work presented here are both NP-complete problems[90]. NP (short for nondeterministic polynomial) is a class of problems for which a potential solution can be verified (though not necessarily found) in polynomial time – the solution can be verified in $\boldsymbol{O}(n^k)$ steps, where n is the complexity of the input and k is a nonnegative integer[91,92]. NP-hard problems are problems (that are not required to be NP) that can be considered at least as hard as other NP problems[93]. An NP-complete problem is both NP and NP-hard[94]. In the case of an internal layout problem, a proposed solution can be checked simply by placing the components of interest into the relevant container and checking for intersections or any violation of constraints. Mathematically speaking, this is actually quite similar to the famous traveling salesman problem, with each component position coarsely representing a city visited by the famous (and rather busy) traveling salesman[95].

The primary practical feature of NP-complete problems relevant to this work is the practical impossibility of discovering the optimal solution without an exhaustive search[96]. Since a solution to one NP-complete problem can, through sets of transformations, be shown to work for any other NP-complete problem,

this would seem to not be the case, but no solution currently exists for any general NP-complete problem outside of brute force. This makes finding such a solution impractical for all but the smallest problems. As a consequence, most research into solving NP-Complete problems has focused on heuristics and approximation techniques that can find near-optimal solutions with greater efficiency[97]. The problem is still significant, however, and as of the time of this writing, the Clay Mathematics Institute is offering a prize of one million dollars to the first person who can prove that there is (or is not) a solution to NP-complete problems[98].

Since most practical internal layout problems contain more than a few parts, and since each part has a host of different positions available, there is little expectation of finding the ideal solution quickly using a direct method. In the absence of a winner of the Clay prize, a more rapid approach involving approximations will be used within this work as well.

5 Research Concerns

Along with the research questions and actions mentioned above, this work must also consider issues that, while not unique to this work, are important in the scope of the research. These issues tend to be more practical in nature and concern the mechanics of the proposed work rather than its theoretical aspects. The following sections are designed to clarify some of the more specific issues that this work will have to address. The options presented here have provided the final framework for the prototype tool created to test the hypothesis. They also provide alternatives for future research that seeks to improve upon the work completed here.

5.1 The Structure of CAD and Internal Layout

The automated nature of this work and the desire for rapid solutions logically relies on computers and their affiliated software. Choosing the correct path for developing a useful tool could, then, become almost as much a study in computer science as in internal layout and aerospace engineering. While the temptation exists to drive the bulk of the work here into the details of optimization and computerized manipulation of data, the author has resisted, preferring to concentrate on creating the most efficient tool for demonstrating the effectiveness of a volumetric sizing methodology.

CAD software provides much of the functional foundation of this work. Even though the full feature set of a CAD package is not required, the basic

elements of CAD are necessary for the virtual arrangement of components within the model, and the inclusion of CAD in later phases of design is essentially assured by its domination of the later phases of modern aircraft design. For the purposes of this work, CAD elements should allow simple automation, accurate placement of internal objects within a designated space, the ability to scale objects parametrically, and a collision model to keep track of intersections between objects and the boundary of the space (or each other).

CAD applications also bring with them access to libraries of standard components. While completely described standard components are not always available for the conceptual phase of the design process, the ability to access standard shapes is a benefit of a fully fleshed CAD environment. Completely described components are also not always desirable from a numerical perspective, as described above, but the tight integration of modern CAD tools with other analytical tools and product lifecycle management (PLM) capabilities are significant benefits modern CAD brings to volumetric sizing. Below is a selection of CAD-related tools considered for this work. Note that not all of these options are fully fleshed CAD applications but are instead components of a CAD or related visualization environments.

5.1.1 CATIA (Dassault Systemes)[99]

CATIA is a robust and relatively well-documented modeler designed as a core component in Dassault Systemes' product lifecycle management toolset. Unfortunately, the program is quite large and therefore slow for the purposes of this work. Two local *CATIA* experts have recently left the institute, steepening the author's learning curve for the software. *CATIA* cannot currently run on local

computer clusters without significant modification to the clusters and a way to reconcile licensing concerns. Also, there is no access to the *CATIA* source code in the event that changes need to be made or features added. It is, however, an effective tool for later portions of the design process and would almost certainly integrate well as part of a future volumetric sizing effort. A sample of internal layout during detail design of a ship using *CATIA* is shown in Figure 19.

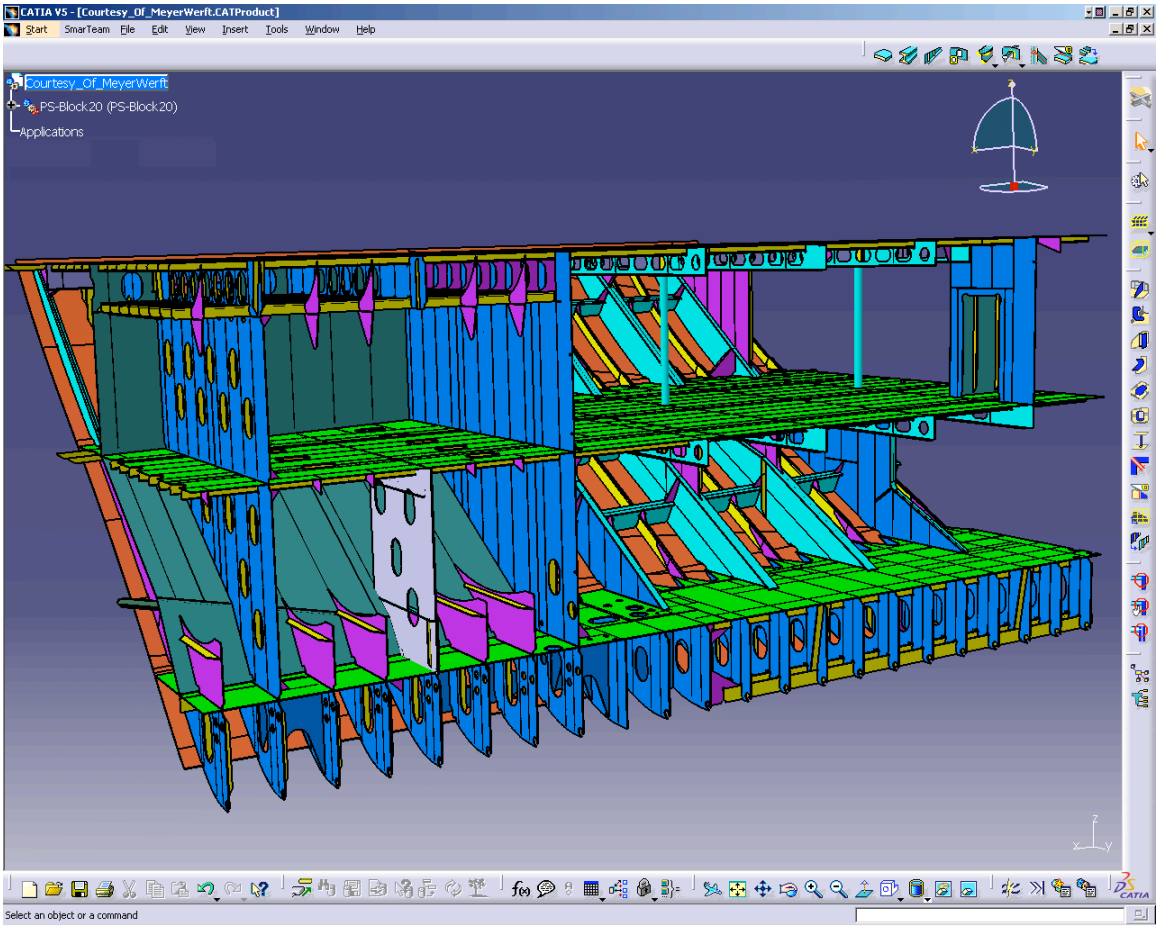


Figure 19: *Ship detail design internal layout in CATIA*[100]

5.1.2 BRL-CAD (US Army Ballistics Research Lab)[101]

BRL-CAD is well-developed, free, and open-source. Another solid modeler, it currently runs on most non-Windows platforms, including Linux and Apple's OSX. It is extremely scriptable with Tcl, and can interface with small C applications. Additionally, *BRL-CAD* requires no graphical user interface (GUI), so the portions of the code important to this work can run without the delay caused by constant re-rendering. It does have limited support through a network of developers and a four-volume instruction manual. While unsupervised modification of the source code is not encouraged, the source is available should manipulation be required. The manual is fairly well developed, but it is better suited as a basic user guide than as a development tool. A sample render from *BRL-CAD* is shown below in Figure 20. Note, however, that this image took five days to render on 48 2.4Ghz Xeon processors[102].

Figure 20: *Complex rendering from BRL-CAD*[103]

5.1.3 OpenCascade[104]

OpenCascade is a set of open-source surface modeling libraries. Applications can be developed for low cost using these libraries, but most applications available that use OpenCascade are commercial products not well suited to this work due to platform and licensing issues. The primary development thrust with OpenCascade seems to be in generating paid integration work for the group that developed the libraries. The set of libraries should run fine on any platform, but there is no free, complete application making use of them. Support is also not free, as it relies on a commercial

enterprise related to the development group. No local expertise exists for OpenCascade, either, so even informal support is unlikely. Two images created with applications based on OpenCascade are presented as Figure 21.

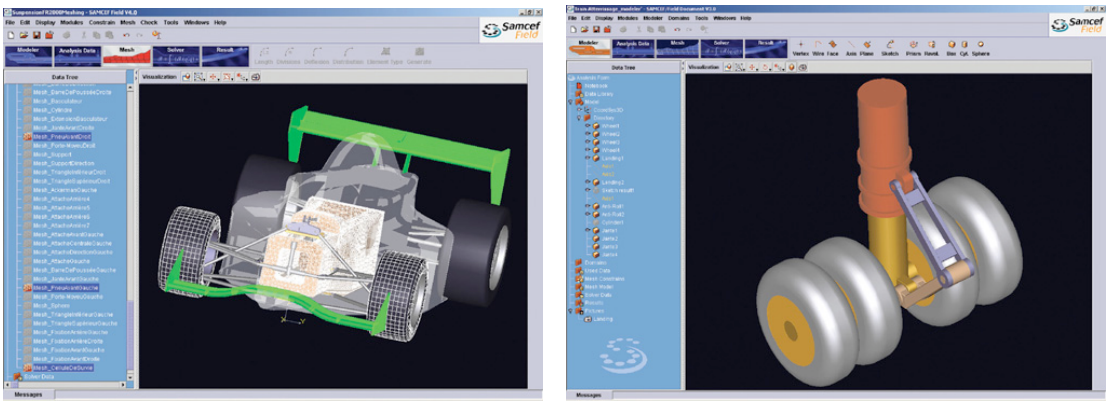


Figure 21: *Car and landing gear modeled using OpenCascade technologies*[105]

5.1.4 GTS (GNU Triangulated Surface Library)[106]

Similar to OpenCascade, GTS is a set of surface modeling components in the form of a library of functions. Free, they can be used on most platforms. Unfortunately, there is not a complete application associated with the library outside of some small utilities, so most coding has to be done from scratch. The installation does come with several small programs that demonstrate several of the more commonly-used functions. Some local expertise exists with application development using GTS as the ASDL-developed *triAero* aerodynamics calculation tool makes significant use of GTS[107]. Unlike OpenCascade, however, the entirety of the project is open source, so licensing is less restrictive. GTS can also run on nearly any Unix-like system, from the BSD-based Mac OSX to most

versions of Linux and on Windows systems with some restrictions. A sample complex surface created in GTS is shown in Figure 22.

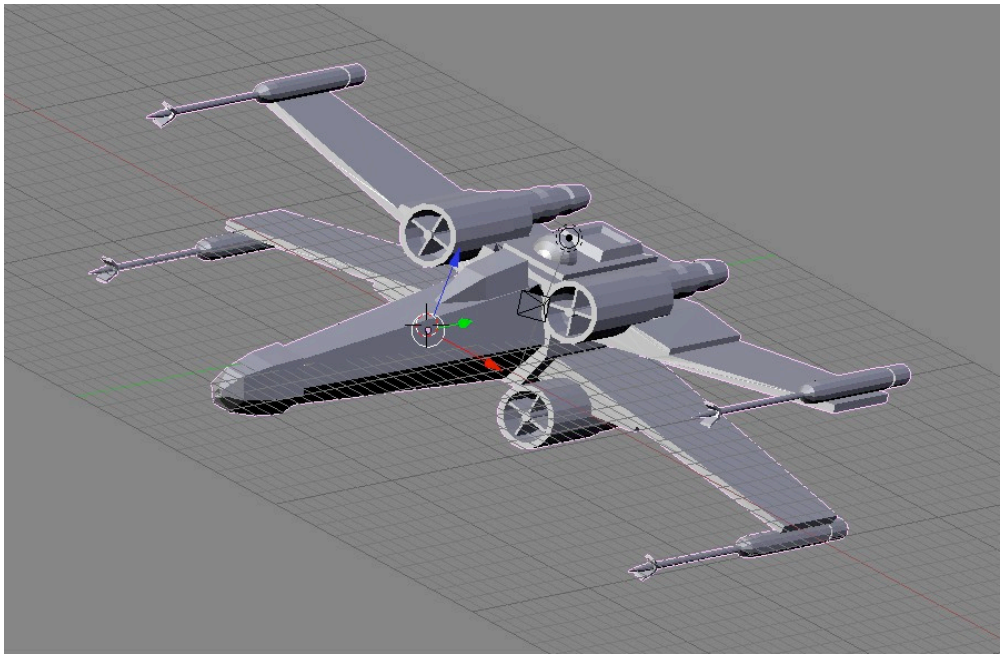


Figure 22: *Complex GTS-generated surface*

5.1.5 Self-Programmed CAD

While a homemade CAD program offers the greatest level of customization, it also requires the most work for the least amount of return. It could, however, run on any system for which it was tailored. If properly optimized, it would be quite fast, as the program would not have to waste time with any unnecessary options. Unfortunately, there would be a significant amount of programming overhead inherent in the creation of such a tool, as even the most basic elements of the program would have to be generated from scratch.

5.1.6 Blender[108]

Blender is an open source 3D graphics creation tool. It is designed to be used by content creators to generate detailed 3D representations for animation, rendering, and other visualization needs. Blender does offer many of the Boolean operations needed to perform intersection checks between components, but it is not really designed for the type of work being explored here. It is, however, quite useful as a visualization tool in concert with other programs and is capable of importing a host of 3D file types, including those used in the final output of this work. Most of the images displayed of the final working tool used in this work were generated within *Blender*.

5.2 Placement Technique

In cooperation with the selected CAD foundation, a volumetric sizing tool needs a reliable, robust technique for arranging components within the designated space. Ideally the tool would be able to handle a wide variety of fit options that can be tailored to the system in question. While conceptually sound, generating a wide variety of fit options to choose from during an analysis cycle is a task beyond the scope of this work. Instead, a single fit methodology will be used to demonstrate the effectiveness of the overall technique.

The actual technique used for placing the components into the available space can be described generally in one of two ways: *arbitrary placement* and *path-based placement*. For arbitrary placement, components are placed without regard

for a path for installation or any manufacturing or maintenance concerns. The advantage to this placement technique is its simplicity and relative speed. Disadvantages are related to manufacturability and maintenance issues. For some purposes, such as missiles or other "maintenance-free" aircraft, these may not be important concerns. For more conventionally operated systems, maintenance and manufacturability are extremely important, as is the ability to remove components for repair or replacement. Unfortunately, these needs only increase the complexity associated with a robust, sophisticated path-based placement system. Further, at the conceptual design phase, details like wire routing and component assembly are still too detailed for consideration because of the additional computational requirements they bring with them.

The notion of ordering also becomes important here. Ordering components in an offline problem in one or two dimensions typically saves a significant amount of time in generating a solution. In three dimensions, ordering the components can also save time, but only in relatively loosely constrained problems. Complex constraints can cause traditional ordering techniques to produce less than optimal results. One extremely common ordering technique is to use size as the criteria – components are placed with the largest going first. In a path-based solution, that would place all of the largest components at one end of the container, an arrangement that could play havoc with center of gravity constraints, among others.

A simple example can seen in Figure 23. Here, four components with a mass of 1 unit each are required to balance on a beam with a larger component with a mass of 4 units. In the case of simultaneous placement, the shapes balance with no problem. For an ordered placement technique using any possible

arrangement, the components will not meet the constraint at every point in the process, though they can meet the constraint at the end of the placement. Two solutions exist: wait until the end of the fit process before calculating constraints, or use a fit process that does not have interim steps.

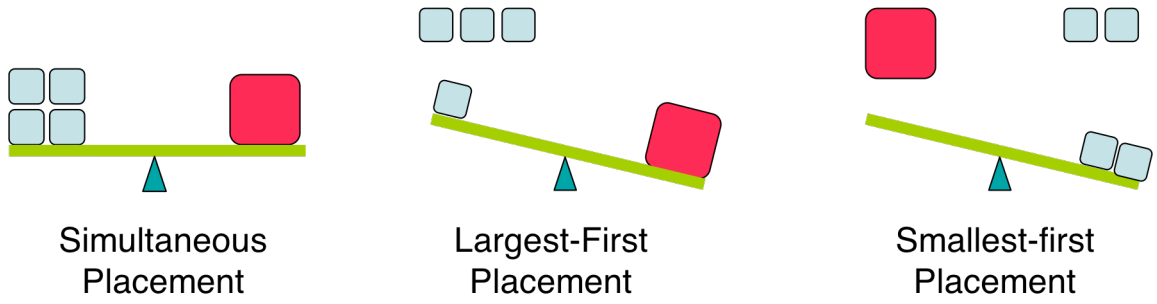


Figure 23: *Possible consequences of poor ordering arrangements*

Waiting until the end of the fit process before calculating constraints solves the problem illustrated above, but the process of ordering components may bias the fits in such a way that none of the resulting arrangements will meet the constraints at the end. In the simple sample above, a largest-first fit criteria might put the large component and three of the small ones on one side of the beam. This will always result in a constraint violation. A smallest-first might evenly distribute the small components about the fulcrum, only to have the large component disrupt the entire arrangement. In contrast, a simultaneous placement technique may also violate constraints, but it saves the computational efforts of ordering the components initially and speeds the evaluation process.

Deciding which path to follow involves the investigation of several techniques, described below:

5.2.1 Drop-Fit:

Components are "dropped", Tetris-like, into the test space and rotated to obtain the best fit. This is a relatively quick path-based placement technique that keeps the combinatorial load minimized. While it should work extremely well for box shapes, it cannot properly fit "hooks", nested solutions, or other unusual shapes. It is also poorly suited to "borderline" solutions – component arrangements that may be very close to optimal yet have some overlap, since it cannot naturally handle component overlap at all. A drop-fit technique does require ordering, however, as placement of later components depends on the placement of the initial components. The ordering process here would be ineffective without extensive knowledge of the system being designed in order to ensure that constraints were not invalidated simply due to a poor choice of component order.

5.2.2 Point-Rotate:

Components are placed at the first available point and rotated until a fit is produced or all possible combinations have been tried and the component is moved to the next available point. This simple method allows accurate fits for a wider variety of component shapes, but it requires more attempts to fit all components and sophisticated collision checks. Without modification, it also cannot easily handle component overlap. The notion of placing at the first available point also implies a need for ordering, as later component placement

will depend on earlier component positions, and ordering issues are similar to those faced above.

5.2.3 Random Placement:

Components are placed randomly with arbitrary rotations and checked for fit. Extremely simple, this technique allows rapid placement and assessment of individual cases. It is also extremely robust – eventually it will find every possible solution. Unfortunately, these advantages lead to a high incidence of cases with multiple collisions, and the process requires many runs to find good cases. Random placement can handle overlap with ease, but that capability can also lead to many poor arrangement groupings. No ordering is required here; in fact, for a purely random placement technique, any ordering would be wasted.

5.2.4 Guided Placement:

Components are restricted to "good" locations, placed one at a time so that no intersections among the component parts occur. This method eliminates most "bad" cases and impossible configurations, speeding convergence when compared to random placement, above. Unfortunately, it requires extensive logic to implement and additional computational complexity. The complicated nature of the component placement causes problems with the stochastic nature of some optimizers. It also has to be carefully designed to allow "almost good" solutions that will be the best available in the event that a good solution is impossible. Otherwise, the placement algorithm will be stuck in an endless loop as it tries to solve an impossible problem. Ordering again becomes an issue with guided placement for reasons similar to those mentioned above, but some simple

ordering by volume may help speed the convergence process through elimination of bad cases earlier in the process.

5.2.5 The Golden Path:

Conceptually similar to a combination of guided placement and drop-fit, the golden path placement requires an open path to the final placement location but allows turns and multiple orientations as components are steered around previously placed items. This technique emulates the actual manufacture of some items, where a path must exist for removal/servicing as well as for initial construction. This assumption is not always required, however, as some vehicles, such as missiles, will have no servicing requirements. This method is best suited to less-dense systems that are farther along in the design process. Ordering adds computational resources and constraint challenges here as well.

5.3 Optimization Technique:

In addition to the large number of possible combinations involved in the location and orientation of components in an internal layout, the design space is riddled with discontinuities and irregular features. Overlapping components, for example, would result in a configuration that is physically impossible, and therefore undesirable. Yet a solution close to a poor solution could be desirable. In fact, since the most compact solutions occur when internal components are

near one another, having an irregularly shaped design space is almost guaranteed, which makes finding an optimal solution more difficult.

The design space in this work is the set of all possible coordinate points for every component under consideration. Each arrangement of these points produces its own set of characteristics: intersection volume between components, intersection volume between components and the container, constraints such as center of gravity, and placement constraints for individual components. The goal then is to explore this design space as thoroughly as possible and discover the arrangement of internal components that best captures the needs of the rest of the design tools.

5.3.1 Localized Optimization

Local optimizers are often the most rapidly considered optimization technique considered for simple tasks. Local optimizers, for the purposes of this work, are optimizers that are designed to find an optimal solution in a space that is relatively well-behaved. In many design problems, only a portion of the design space is well-behaved – at some point elements of the design space can get more difficult to analyze. The triggers that take a space from being well-behaved to being difficult to analyze are dependent on the optimizer. Many optimizers, for example, assume that the space is continuous, that if a variable is followed from one extreme to another that there will be no "holes" where a solution cannot be found. Others assume that functions will be monotonic, either increasing or decreasing in a single direction with no "hills" or "valleys" on the way to the best solution. Many of these optimizers are considered path-building optimizers,

as they will tend to follow a set path, much as water flows down a mountain to a stream at the bottom.

One great quality of local optimizers is their capability to rapidly find the best solution in a relatively confined space. Their computational requirements, while not usually trivial, are usually modest in comparison to the process being optimized, so they are extremely efficient in certain types of searches. They are not efficient, however, in spaces that are more difficult to describe. In the case of many first-order techniques, which require derivatives to describe the basic topography of the space, a design space that contains areas where no derivative exists may cause the optimizer to fail. Even in cases where the optimizer can function perfectly well, it may be "fooled" by local optima that mask the design space's real optimum solution.

For example, if an analyst wanted to find the highest point on the Earth using a local optimizer, the result would most likely be failure. Starting from practically anywhere, for example, a local optimizer would lead up to the top of the first nearby hill. In a rare case, the result might lead to the top of a small mountain in a nearby community. Only if the search were initiated near the true optimum would the solution actually reach that summit. For problems with a great deal of complexity and depth, such as this work, another technique is needed.

5.3.2 Domain Spanning Optimization

Traditional path-building optimization techniques are typically ill suited for design spaces with a large number of local optima. In the example above, every local hill becomes a local optima, so a simple path-building optimizer will

not find the highest point with any significant degree of certainty. Similarly, problems with internal layout are complicated and have many arrangements that may not lead to the best answer with a path-building optimizer. In many optimization problems, the position of the point where the search is initiated makes a significant difference in the quality of the result. Just as starting a search on Mt. Everest will increase the chance of finding its summit, starting an internal layout problem with an arrangement close to the true optimum increases the chance that it will be found[vii].

One approach to solving this problem is to simply use a large number of starting points. Unfortunately, without a thorough understanding of the design space, selecting the best starting point can be difficult. This is traditionally overcome by utilizing a large number of starting points, selected either in a pattern or randomly. This technique will eventually lead to a solution, but the number of tries may be prohibitive when looking for an efficient answer.

This notion of trying many points is at the heart of domain spanning techniques. Rather than building a path to the best answer based on an initial point, domain spanning techniques attempt to cover the entire design space with a web of possible solutions, refining the search to exclude poor choices and finally obtain the best answer. The most basic domain spanning technique is also the simplest. Simply trying every possibility in a design space will guarantee the best solution, but the time and effort required often precludes such an attempt.

[vii] The peak of Mount Everest is considered the highest point above sea level. While there are other mountains that can be considered the tallest, for the purposes of this example, Everest is most appropriate. http://en.wikipedia.org/wiki/Mt._Everest

Indeed, that choice has been explored for this work and immediately discarded as impractical based on the mathematical exploration in Chapter 4.

Other search techniques have also been developed. Pattern searches rely on a grid of points that are sufficiently close to capture a solution close to the optimum while being sufficiently sparse to reduce runtimes to a reasonable value. Pattern searches are the simplest choice for optimizing problems containing continuous variables, as the notion of running every point is nonsensical for problems containing an infinite number of points.

Within the past thirty or so years, more efficient domain-spanning techniques have become possible with developments in computer technology and algorithm development. Search algorithms such as Genetic Algorithms (GA), Simulated Annealing (SA), Branch-and-Bound (BB) and others have allowed the relatively rapid exploration of large solution spaces featuring unusual shapes, discontinuities, and other features that would potentially render path-building searches unsuitable.

As Cagan, et al. detail, layout problems have been approached with both GA and SA techniques, as well as with hybrid approaches and modified gradient-based techniques[109]. Pattern searches, an outgrowth of the originals from many years ago, have also been applied with good results. While at a significant disadvantage from the perspective of viewing the overall design space, gradient-based solutions do offer guides to possible improved solutions. As components are placed in an internal layout, moving them closer together tends to improve the quality of the solution, and gradient-based optimizers tend to drive the components closer together – and to push them apart in the event of overlap. GA, and to a lesser extent, SA, optimizers rely more on the clustering of

good solutions around randomly selected solution points, and they have also been applied with success to problems involving internal layout[110].

While capable of spanning large spaces, these optimizers may not have sufficiently efficient guidance to derive an acceptable solution in a reasonable amount of time. Tracking some of the possible outcomes, this inefficiency is easy to observe – domain-spanning solutions spend a good deal of time inspecting possible solutions that would be discarded almost immediately by an expert user doing the work by hand. This drives the need for a more intelligent optimizer for complex layout problems.

5.3.3 Branch and Bound

Branch and Bound (BB) techniques have been used on a variety of NP-Complete problems, so applicability to this problem is almost guaranteed in at least a limited capacity[111]. It is initially attractive because of its regular, predictable nature. The BB technique makes regular divisions of the design space, subdividing it until a limit is reached. These divisions start with a single function evaluation and branch into subregions, until the design space is completely covered with sub-branches at the resolution desired by the user[112]. The problem evaluated at the root is now propagated through the branches in an attempt to find the best solution.

The "bound" portion of the technique's name comes from the method used to eliminate portions of the design space. At the initiation of the problem, a lower and upper bound are created for the space (or at least estimated). For each sub-problem encountered on a branch, the result of the evaluation of that node is compared to the lower and upper bounds of the rest of the space. For a

minimization problem, if the lower bound of a node is greater than the upper bound of the best existing solution, the branch beginning at that node can be "pruned", and all portions of the space below that node can now be safely skipped.

The ability to construct the divisions of the problem into smaller subproblems and to compute the upper and lower bounds are key to this method, though accomplishing this is a significant source of difficulty in using a BB[113]. In an idealized BB arrangement, the space should be separated into a large region of poor solutions and a smaller region of the best solutions. This allows the technique to rapidly eliminate a large portion of the space, reducing the overall combinatorial load. While this is an attractive feature of the BB method, arranging the problem to best suit the solution requires a deeper analysis than is practical here. Even the seemingly simple task of calculating a lower bound effectively requires an analysis beyond the scope of this work.

5.3.4 Genetic Algorithms

Genetic algorithms borrow from the evolutionary analogy as much as possible. Design variables are discretized and combined into a long string of possible combinations called the "DNA" of the design space. Traditional genetic algorithms (GA) rely on three primary operators to explore the solution space: replication, mutation and crossover. These three operators function over a series of "generations", sets of possible solutions grouped together and run simultaneously. Replication in a GA continues the "genes" of the most successful solutions by repeating them from one generation to the next. This behavior is

based on the initial assumption that the best solution will probably be somewhat near other "good" solutions.

The mutation operator randomly changes portions of the DNA string of a set of input variables according to a set of rules predetermined by the optimizer. This should send the optimizer to a more desirable portion of the design space and is done in an attempt to keep the optimization process from being bogged down by local optima. The crossover operation attempts to similarly explore new portions of the solution space through a combination of the qualities of two existing sets of inputs. The "offspring" from the two produces another set of inputs with similar (though not identical) characteristics to the "parents" in an attempt to explore nearby areas of the solution space. It is the combination of these operators that differentiate the GA from a truly random search.

5.3.5 Random and Grid Searches

A random search would be simple to implement in this type of methodology. With such a large area to search, a random search would provide a relatively good distribution of points over the design space. Unfortunately, with no further intelligence to focus the search in areas likely to provide good results, a basic random search is unlikely to provide good results without some other component to improve searches locally. This would generally produce a poor convergence rate, as NP problems in general do not respond well to random searches.

A grid search was seen as an alternative to a random search. Instead of using a collection of random points, a grid search follows a pattern to find points to evaluate. Traditionally, these patterns have tended to be basic square grids, as

this method guarantees a certain regularity of coverage not found in a number of domain-spanning search techniques. Unfortunately, apart from this guarantee of regular coverage, a grid search offers little in terms of performance advantages since it is unable to dynamically adjust its search resolution to match the probability of finding an improved solution.

5.3.6 Simulated Annealing

Simulated annealing is an optimization scheme that is conceptually inspired by the process of annealing metal. As a metal is heated, it becomes flexible; as it is worked while it cools, it loses flexibility and gains strength. Finally, the metal is no longer malleable and is in its final shape, with strength far greater than the initial, unannealed sample. In simulated annealing, a set of prospective solutions is generated and then perturbed. If these perturbations result in better solutions, the results are kept. If the perturbations result in a worse solution, then the solutions are only kept if a particular probability check is passed. It is in this probability check that the annealing analogy comes into play.

The probability check in simulated annealing is related to a "temperature" of the solution space, and it can be represented mathematically in a variety of ways. At the beginning of the search, the temperature is set high, and the probability that a worse solution will be accepted stays high. As groups of solutions are tested, the temperature drops, decreasing the probability that answers worse than the current answer will be accepted. The basic premise behind the use of simulated annealing is that accepting a worse solution

occasionally will allow the optimizer to "jump out" of local optima and more effectively find the best solution.

One feature of simulated annealing that connects it somewhat to gradient-based techniques is its notion of walking. Potential solutions are chosen within relatively close proximity to existing points, and future attempts will also tend to cluster around the initial point unless the design space encourages large movements.

5.3.7 The Moreau Operator, a Hybrid Approach

In the case of an internal layout, none of these above solutions will predictably explore the parts of the solution space which are intuitively "better" than that found by the existing inputs. One solution is a form of "genetic manipulation". The Moreau operator, created as part of this work, will be considered as one technique for accomplishing the desired manipulation.

Dr. Moreau is a fictional character in a novel by H.G. Wells[114]. Working in secret on an isolated island, he mixes the characteristics of beasts with those of humans to create hybrids – he makes the animals into something they are not by forcing a change. In the same manner, the DNA of inputs into a GA can be manipulated to change a particular input group into a more desirable set of values.

While the fictional Dr. Moreau faced unpredictable behavior in his progeny, modifications to DNA strings in a GA can result in predictably better solutions. In a tightly fit internal layout, one with few open spaces, the vast majority of randomly placed sets of components will result in overlap – a physically impossible condition. Overlap, while difficult to rectify within the

traditional bounds of a GA, can be easily solved with a gradient-based system. Internal components can be encouraged to be as close as possible without overlap. This type of modification, of course, changes the values for input values originally selected by the GA. The new values can then be run through the same assessments as the other members of the GA population. There is also the risk that reducing overlap between two components will only increase overlap between one of them and another.

To be effective, the hybrid mutation technique must still allow nonconvergent designs that may be close to the optimal solution, but it also needs to do its best to improve designs that are close to convergence. Design of an optimizer that can strike an effective balance between accuracy and speed while allowing an effective exploration of all possible solutions is of high priority for this work.

5.4 Organizational Arrangements

Once a fit methodology is in place, an organizational scheme is necessary to choose the order of components for packing within the designated space. Many problems in traditional bin packing are on-line problems. As Csirik, et. al. explain, this refers to problems "that must permanently assign each item in turn to a bin without knowing anything about the sizes or numbers of additional items"[115]. The popular video game Tetris is nearly an on-line problem. Users are presented with only the next shape in addition to the one they are placing.

Fortunately, for volumetric sizing applications, the number and size of every item is known within a particular bound of uncertainty for a particular instance of the problem.

This means that many of the algorithms used for on-line bin packing problems will not apply particularly well for the problems inherent in volumetric sizing applications. This does not mean that every element of the solutions is without usefulness, however, and elements of several of these algorithms will find themselves in the final solution approach used in this work.

While volumetric sizing is inherently an off-line bin packing problem, constraints added to the basic volumetric problem bring back some of the same problems encountered in more traditional on-line problems. One facet of on-line problems that increases their difficulty is that they force any algorithm to place an object into a location that may not be the optimal placement when the problem is completed. With a fully off-line problem, these problems are overcome by the complete knowledge of the objects to be placed – a good algorithm will consider all possibilities in an off-line problem while the online algorithm must make do with what it is given, when it is given it – a very handy capability in a production line environment, for example.

With constraints in an off-line problem, the algorithm is again forced to "make do" with what it has in order to satisfy requirements other than those strictly formed by the geometries of the individual components. A stringent center of gravity requirement on an airframe, for example, may completely preclude a large number of otherwise satisfactory solutions by virtue of one or more heavy objects that throw the solution out-of-bounds. This is very close

conceptually to problems an on-line algorithm would have with placing items out-of-order in response to the unpredictability of the problem.

A selection of algorithms is presented below to help explore some of the possible solution techniques for this work. Because of the NP-hard nature of the bin packing problem, none of the algorithms presented offers a notable performance advantage over the others. Differences are primarily described in terms of "performance ratio"[116]. This ratio, as described by Kenyon, after Johnson, is the length of time for convergence of the algorithm of concern compared to that of an ideal algorithm[viii,117]. This is done versus a worst-case or random-case arrangement of objects. An extremely efficient algorithm can be designed for a best-case scenario – essentially an off-line problem, but many of these algorithms will perform poorly with any other arrangement of objects. The goal, then, is to design an algorithm that performs well with any arrangement of components. Most commonly-studied on-line algorithms range in performance from a ratio of around 1.54, where a lower bound has been determined (though its exact value has been moved a number of times), to a ratio of 2.0 – that of the popular next-fit algorithm[118]. With the depth of combinatorial complexity present in the basic internal layout problem, moving from a factor of 1.54 to a factor of 2.0 is not much of a problem.

[viii] Kenyon's expression is: $C(A) = \limsup_{OPT(L)\to\infty} \frac{A(L)}{OPT(L)}$. Where C(A) is the performance ratio, OPT is the (unknown) optimal algorithm, L is a list, and A(L) is the number of bins used by algorithm A on L.

5.4.1 Sum-of-Squares:[119]

This algorithm is conceptually similar to many other on-line packing algorithms. It uses a sum-of-squares calculation for determining the best location for placement of each object. Since the solution is applied to a one-dimensional problem (the value of the sum) rather than the three-dimensional one (the actual location), this results in a simple, effective measure of the quality of fit for a particular item. The one-dimensionality also precludes multiple arrangements in a single bin with the same value. Csirik, et. al. also present some alternative algorithms, including an off-line sum-of-squares algorithm with good results[120].

5.4.2 Best-Fit with Random Order:[121]

The best-fit algorithm is the best known of the bin packing algorithms. Given a series of bins, it places items in the fullest bin into which they will fit. It is a relatively quick algorithm, with a performance ratio of 1.7 in the worst case. Practically speaking, where the worst case is no more likely than any other case, it performs extremely well. For the purposes of this work, best-fit would be used to place components into the tightest space remaining in a discretized container.

5.4.3 First-Fit:[122]

First-fit is similar to best-fit in its simplicity, though it tends to be less efficient. When a new object is introduced, the first-fit algorithm places it in the first bin into which it will fit. This omits any of the judgment inherent in best-fit, and violates many of the common-sense "rules" used by human loaders of cargo and other objects. In this work, the inefficiencies inherent in first-fit algorithm may outweigh the simplicity, rendering it less effective than other algorithms.

5.4.4 Next-Fit:[123]

Next-fit is again similar to first-fit, but it does not store any but the last bin, making a new one in the event an object will not fit into the current bin. This reduces performance significantly while minimizing memory requirements for the algorithm. The distinction between next-fit and first-fit may be insignificant in the context of this work, as there is typically only a single bin to use, so its usefulness is suspect.

5.5 On Computational Efficiency

One of the most important contributors to the overall speed of any volumetric sizing tool is the efficiency of the technique used to implement it. This efficiency, in turn, is affected by the application framework chosen and the hardware on which it runs. Selecting the correct toolset then becomes crucial. An optimization scheme that incorporates some level of intelligence to rapidly eliminate impossible cases also contributes to a more efficient solution process. Another concern is that of potentially running the tool on multiple computers. With much greater performance than any single machine, parallel processing offers enormous potential – if the software tools chosen will work properly.

5.5.1 Connecting the Parts

Selecting the best technique for connecting the various components of this work into a single cohesive demonstration tool involves numerous trades. The programs that are easiest to program do not always offer the best compatibility with an assortment of operating systems and hardware platforms. The most primitive systems, relying on coding in C, Fortran, Java, or some other traditional programming language, may be difficult to program, requiring a great deal of time. The scripting languages, such as Perl, Python, or Tcl may not offer sufficient power. More sophisticated integration applications carry their own sets of problems, from difficulties inserting volumetric sizing into a framework designed for other applications to issues of licensing and platform dependence.

5.5.2 Discretizing the Container

Several commercial codes used for loading shipping containers and/or trucks make use of a relatively simple technique for reducing computational complexity in their solutions. By breaking down the container into a series of small cubes and then forming any internal components of identically sized cubes, the authors are able to reduce a complex problem to one of far fewer possible combinations.

This simplification works well for shipping containers, as they are rectangular and easily discretized in this manner. The contents of these containers also tend to be boxes, which can be almost perfectly discretized, or crates, which are also easily discretized. Odd shapes and liquids can be accommodated as well – their boundaries are merely extended to fill a series of unit cubes.

Unfortunately, aircraft are rarely rectangular, nor are many of their parts. While some discretization necessarily takes place at the limit of the computer's ability to resolve differences between two numbers, this is at quite a smaller level than that used to discretize cargo containers. The discretization practiced by cargo companies is also at odds with the standard way most modern CAD programs are put together. Since part of the goal of this work is to make us of a wide variety of shape types – including standard shapes from a CAD library, it seems more logical to keep the fundamental layer of the process at as fine a level of detail as possible.

The problems with such a coarse representation are easy to identify. Figure 24 shows four common techniques for representing an object in a space. Object A is represented analytically. It is, for all intents and purposes, the object itself that will now be manipulated. Unfortunately, this representation is difficult to express in ways a digital computer can understand, particularly if the shape is complex. Shape B in the upper right represents the way shapes are often represented in cargo loading software. The coarse divisions make computation easy, but the result is an enormous amount of wasted space. Shape C, in the lower left, uses a single bounding box to approximate the shape in an area with finer resolution, which helps reduce the amount of space wasted. Object D, in the lower right, however, wastes the least space while maintaining a high resolution. This is the technique used in this research.

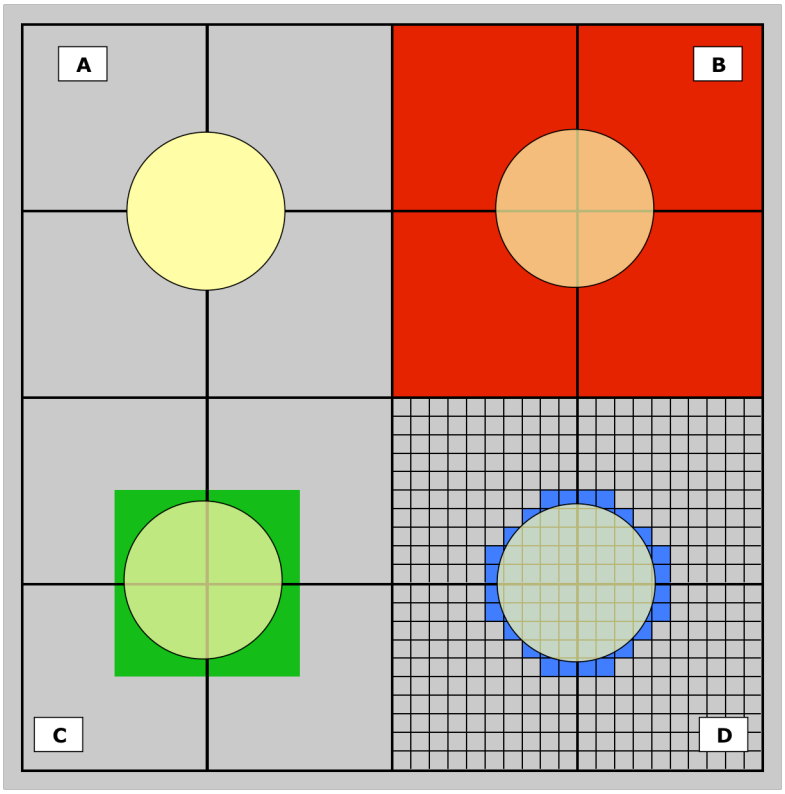


Figure 24: *Common shape representation techniques*

The boxed volume equivalent (BVE) concept mentioned earlier also attempts to gain some of the advantages of cubic simplification while allowing finer detail later in the process. It also allows placement at finer levels of detail than that afforded by cubic simplification.

A practical application of cubic simplification that can be added to this work at a later date involves modifying the simplified volume equivalents (SVE) used in this work. SVEs are formed by reducing the number of triangles making up a surface or using primitive shapes such as spheres, cones, rectangular boxes, and the like to represent more complex components. Future work could consider using a cubic simplification system to form the SVEs. This broader application is beyond the current scope of this work.

Some exploration is done in this work on the possibilities of coarsening and discretization. The initial resolution chosen for this work equates to approximately a tenth of an inch position error on an aircraft the size of a Boeing 777[ix,124]. A dynamic resolution switch was also added to portions of the code to explore performance differences at various resolutions. This is explored in greater detail in a later section.

5.5.1 Algorithm Choices

As mentioned elsewhere in this work, the choice of algorithm for sorting components within the provided space is a balance between raw speed, ability to find a solution for any arbitrary set of components, and the ability to find a solution for any arbitrary set of constraints. The features of any particular algorithm will often be at odds with one or more of these goals.

At the very least, raw speed and ability to handle constraints are features that head in opposite directions. Fast algorithms prefer simple calculations and minimalist constructions, yet constraints add numerous complexities to the entire problem. The bin packing literature has also demonstrated that neither the fastest nor the most complex algorithm is necessarily likely to produce the best results for every set of components. A large number of well-regarded bin packing algorithms are also limited at exercises involving more than one dimension. The three dimensions required for volumetric sizing add

[ix] A box large enough to fit a Boeing 777 would need to be approximately 210ft by 200ft by 61ft. Discretizing a box of that size into 65535 sections in each dimension results in a set of rectangular boxes that are less than a twentieth of an inch in the longest dimension. Even if the plane were placed in the box in the least favorable way, the largest dimension of any of the boxes would be less than a tenth of an inch.

tremendously to the complexity involved in bin packing and add to the combinatorial volume – again providing goals with opposite solutions. The solution, then, is to find an algorithm that straddles the Pareto front of combinatorial sets and complexity; of constraints and simplicities.

6 Tool Selection and Problem Construction

In the case of this work, many tools possessed part of the capability required to demonstrate volumetric sizing of aircraft, but no single tool possessed every quality required for the job. A change of tools in the middle of the process can also cause delays. The structure of the problem itself is also important to the result. While the initial plan was to create an entire sizing environment that encompassed volumetric components, the problem has been narrowed by necessity, and the focus concentrated on those aspects of the sizing process that are being significantly changed. Indeed, if volumetrics are to be incorporated into a host of new sizing protocols, keeping this portion as segregated as possible can show benefit later. The goal is to incorporate volumetric sizing, not just with traditional sizing tools, but also with more advanced techniques, such as the power-based sizing process proposed by Nam[125].

6.1 Tool Selection

The initial toolset selected for this work was based on the US Army Ballistic Research Lab's *BRL-CAD*. However, the problems associated with a weakly-supported CAD platform were eventually found to overwhelm the advantages that it brought to the work. *BRL-CAD's* use of analytical primitives for most of its modeling, while attractive from a numerical perspective, became a liability when intersection analysis was needed, as *BRL-CAD's* collision detection

was not as robust as initially estimated. Speed was also an issue, as collisions in *BRL-CAD* took several seconds to detect. It was also slow to render. The image in Figure 20 took a 48-processor system five days to render.

The GNU Triangulated Surface (GTS) library was eventually selected to replace *BRL-CAD* as the structural foundation for this work. While bringing increased overhead in terms of programming, GTS does offer a robust collision model and rapid calculations. It also offers built-in shape simplification that further enhances computational efficiency.

Since the decision to use GTS was made, and after several thousand lines of code had been written, suggestions were made about the usefulness of tools such as *PACELAB*[126] and Technosoft's AML[127]. Both tools were initially considered in the initial downselecting of frameworks. Several users of these packages initially consulted for this work felt that neither tool offered what was needed to completely explore volumetric sizing. Both of these solutions would offer much of the CAD functionality needed for this work, and integration with optimizers would be more easily arranged, but speed and a fast, robust collision model were considered to be better found elsewhere.

This does not mean that either package is completely unsuited to volumetric sizing. Indeed, at the time of this writing, Technosoft is actively working with a different aspect of volumetric optimization using AML[128]. A sample design created in Technosoft's *AMRaven* aircraft design software is shown below in Figure 25.

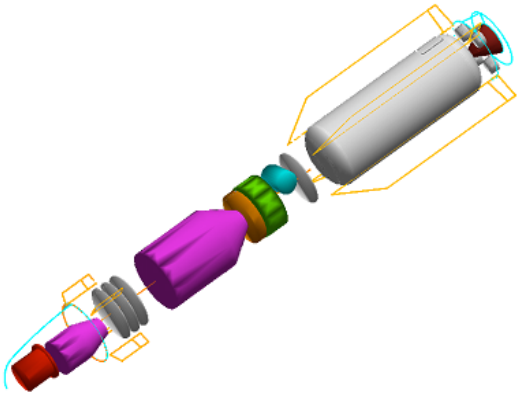


Figure 25: *Missile internal layout in AMRaven*

In retrospect, the current version of Technosoft's AML would have been able to perform much of the mechanical work needed for this project with far less programming overhead, but computational efficiency is still a concern, as is licensing for use on clustered computers. *PACELAB* still seems to be better suited to applications further along in the design process. One of PACE's specialties is cabin layout, though the level of detail used with their tool is better suited to work farther along in the design process. A sample screen capture from *PACELAB* is shown in Figure 26.

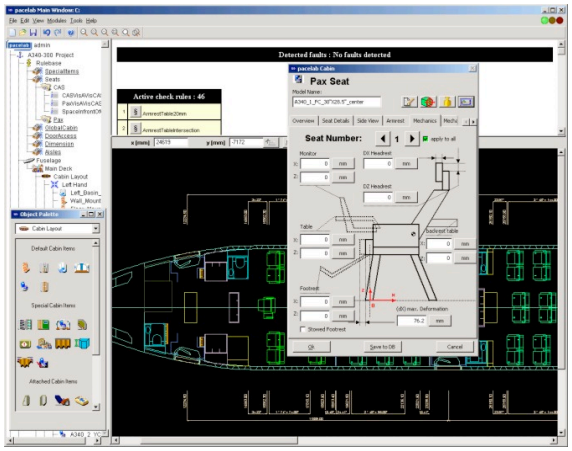


Figure 26: *PACELAB cabin layout screenshot*

6.2 GTS: The Gnu Triangulated Surface Library

Initially, the additional programming needed to work with GTS seemed to outweigh its advantages. After *BRL-CAD* was found to be unsuitable, however, the particular qualities of GTS became more attractive. The robust collision detection was a particular benefit, as it can operate on surfaces with any arbitrary level of detail (limited only by the capabilities of the host computing platform). GTS has not been without its own issues, however, and fully integrating the rest of the toolset within the confines of GTS' strict C architecture has been a particular challenge.

GTS is implemented as a library of defined functions. It works in conjunction with GLIB, a low-level core library used by a number of other packages, including GTK (which underpins the popular open source *GIMP* image manipulation software) and GNOME[129]. According to its online documentation, GLIB provides "data structure handling for C, portability wrappers, and interfaces for such runtime functionality as an event loop, threads,

dynamic loading, and an object system"[130]. GTS primarily makes use of GLIB's capabilities for dealing with trees and hash lists in order to better maintain its surfaces. GTS also uses some structural components within GLIB that help encourage object-oriented features in what is not an object-oriented language.

The primary challenge with GTS lies with its documentation. While comprehensive in terms of addressing every available function, it is lacking in practical examples. The provided sample code was far more useful at explaining the use of specific functions than the online documentation. While usability issues were eventually overcome through a large amount of "elbow grease" and the assistance of more experienced programmers, GTS still produced memory leaks within elements of its bounding box modules. These leaks were eventually traced to function calls from within GLIB. While these memory leaks were reduced by the move from GLIB 1.x to GLIB 2.x, sufficient leaks still exist to cause problems with long runs[x].

GLIB was also found to be the source of a problem discovered late in the final test cases. When identically-shaped components are compared through any of GTS' Boolean operations, and segments of triangles being compared are coplanar, the calculations produce an error, crashing GTS and any supporting code running it. The problem is caused by the interaction of GTS and GLIB functions during what is normally an extremely rare occurrence. Unfortunately, the code written for this work produces this situation often enough for the error to become troublesome. The best solution so far has been to ensure that

[x] This generally applies only to runs longer than three days or so, depending on the platform used and hardware capabilities. Normally, the memory leaks will cause the computer to bump into the maximum memory allowed for a single application, and the next attempt to read more memory will crash the entire program.

components being studied are not identical. More details on the problem and the solution used to overcome it are explored in Appendix B.

6.3 Optimizers and Integrators

Currently, the tool makes use of two optimization techniques. A domain-spanning optimizer generates most of the sample points evaluated in the process of finding an acceptable solution. A local optimizer is then called upon to "tweak" a sample of the results in an attempt to more rapidly converge on potential nearby solutions. This combination of optimization techniques combines the best qualities of both types of optimization techniques in an attempt to more quickly find the best internal layout arrangement.

Local optimizers, for the purposes of this work, are optimizers that are designed to find an optimal solution in a space that is relatively small and well behaved in an imprecise, non-mathematical sense. In many design problems, only a portion of the design space is well behaved – at some point elements of the design space can become more difficult to analyze. The triggers that take a space from "well behaved" to "difficult to analyze" depend on the optimizer being used. Many optimizers, for example, assume that the space is continuous; that if a variable is followed from one extreme to another, there will be no "holes" where a solution cannot be found. If there are holes, the optimizer will not function properly. Other optimizers assume that functions will be monotonic, either increasing or decreasing in a single direction with no "hills" or "valleys" in the way. Many of these optimizers are considered path-building optimizers, as

they will tend to follow a set path, much as water flows down a mountain to a stream at the bottom.

One positive quality of local optimizers is their capability to rapidly find the best solution in a relatively confined space. Their computational requirements, while not usually trivial, are usually modest in comparison to the process being optimized, so they are extremely efficient in certain types of searches. Where they are not often efficient is in spaces that are more difficult to describe. In the case of many first-order techniques, which require derivatives to describe the basic topography of the space, a design space that contains areas where no derivative exists may cause the optimizer to fail. Even in cases where the optimizer can function perfectly well, it may be "fooled" by local optima that mask the design space's real optimum solution.

Domain-spanning optimizers are designed to operate over an entire function domain. In this case, the domain consists of every possible combination of location and orientation for all of the components being considered in the problem. This can be a large space, and it is a rare problem that does not have a complicating feature when it reaches this level of complexity. In this case, the relatively modest discretization of 65,535 location possibilities in each direction plus 180 whole degrees of rotation in each direction provides 1.64e21 possible positions for each component in the test problem[xi]. This is reduced somewhat by the restriction that the components stay in the actual fuselage space, but it is then enlarged by any additional components.

[xi] $65{,}535^3 * 180^3 = 1.64e21$ When this is applied to a large set of components, the results are compounded dramatically.

As this work is a demonstration of the possibility of using volumetric sizing and not exclusively an exploration of the best tools for use in every step of the process, the selection of the best optimization scheme was not the main thrust of the research. While the selection of an effective and robust domain-spanning optimizer was important, an exploration of which specific optimizer is the best is left for future work. In this case, the selection of a genetic algorithm did not seem to impose a significant disadvantage over any other optimization technique. As a benefit, the basic structure of the data used to track the component and fuselage surfaces was found to be extremely well-suited to a GA. This does not, however, mean that the GA was the only tool considered.

6.3.1 Simulated Annealing and Internal Layout

At first, a simulated annealing (SA) approach for the domain-spanning optimization was suggested by the work of Jang and Rhee, who used the method to good effect with layout of a small submarine, but attempts to integrate a SA optimization scheme into the code proved troublesome[131,xii,132]. Several of the qualities of SA seemed particularly appropriate for solving internal layout problems. The selection of increasingly close solutions as well as the tendency to stabilize with time, yet still find at least a local optimum were quite attractive. Unfortunately, the implementation of an SA tool was not as trouble-free as anticipated. The structure of the coordinate points used in the study conflicted with the structure of the SA tool under consideration, so another optimization technique was used instead.

[xii] The simulated annealing tool in question is part of the GNU Scientific Library, a C library of common scientific tools.

6.3.2 Genetic Algorithms

Genetic algorithms seem to be favored as much for their conceptual simplicity and familiarity as for their effectiveness. VanDerplatts, in fact, derides GAs as optimizers, finding them lacking on several counts[133]. Fortunately, several of the characteristics of GAs that VanDerplatts finds to be disadvantages turn out to be beneficial for this work.

One of his largest complaints centers around the fundamentally discrete nature of genetic algorithms. Most design problems exist in continuous space, he argues, while genetic algorithms by their nature require discrete variables in order to operate properly[134]. For the purposes of this work, however, this disadvantage is turned around, as the process of dividing any container space into smaller portions makes all three dimensions a set of discrete values. Similarly, rotations can be represented as discrete angles. As mentioned above, this approach was mandated by the combinatorial difficulties associated with internal layout problems in general. This makes representing position in a GA relatively straightforward.

The division of the problem into a set of components within a container also serves the characteristics of genetic algorithms. Each component has a set of coordinates associated with it, three each for position and rotation. These become elements of each design's "DNA". In the initial sample problem, for example, 20 components each have six elements. These combine to make 120 individual pieces of "DNA". These can then be easily manipulated through the traditional GA tools without the use of binary conversion – a common technique for discretizing design variables in traditional GAs.

The genetic algorithm used in this code is relatively simple and coded from scratch. Rather than making use of binary conversions of design variables, this GA manipulates the individual variables both directly (through the crossover and mutation operations) and indirectly (through random point generation in an external function for new points and the initial population).

The traditional GA operations remain intact in the version of the technique used in this work. Crossover and mutation operators operate on randomly selected sections of each design point. The reproduction operation is handled through a list of the best population members that is carried over from generation to generation. The Moreau operator, described below, rounds out the basic operations within the GA.

The whole framework is supported by component and container surfaces using the GTS library. The surfaces themselves are read in from external files. User-defined component information, such as individual component center of gravity and rotation limits, are also read in from external text files.

6.3.3 Other Potential Search Methods

A completely random search would be easy to implement in this type of environment, and it was for early investigations. The random search was predicted to produce an extremely slow convergence rate, and this was, in fact, the case. Leaving a single computer several days to search for a solution to the early sample problem resulted in a slow rate of convergence and no acceptable answer.

A restricted random search, as explored in detail in a later section, was capable of a relatively rapid convergence of the test case, but this was with no

constraints and a relatively sparse component arrangement. It also relied on an ordering structure that might not allow cases to converge after a poor initial component placement.

Branch and Bound (BB) techniques were also suggested as an option for finding a solution rapidly. BB has been used on a variety of NP-complete problems, so applicability to this problem is almost guaranteed. Good performance, however, is not. According to Clausen, BB techniques require specific tailoring to the particular nature of a problem[135]. Black also suggests that BB is not appropriate if solutions are similar or if improvements are only found gradually[136]. This was found to be the case in initial explorations of the design space. While this is a problem that could almost certainly be overcome with time, the needs of this effort suggested a technique with greater robustness, particularly in the context of constraints. While BB was not used for the large-scale global optimization, a variant of the technique is used by GTS for its internal collision calculations. The use of bounding boxes and trees by GTS is quite effective and robust for simple intersection calculations. However, branch and bound was believed to require too much a-priori knowledge about each problem to make an effective tool for general problems, so its use is left to future efforts[137].

6.3.4 Implementing the Moreau Operator

The selection of a GA as the primary optimization scheme also benefited the Moreau operator. While the basic elements of the Moreau could be applied to nearly any domain-spanning optimizer, the basic structure of a GA was extremely amenable to the inclusion of modifications to the potential solutions.

The GA used in this work keeps track of the "best" solutions as the optimization progresses. The Moreau operator first selects these best samples and performs simple manipulations on them in an attempt to "nudge" components away from each other and the fuselage in the event of an overlap. If the number of Moreau operations exceeds that of stored cases, the operator randomly chooses from the remaining members of the population until the number of operations is satisfied. An internal switch also gives the Moreau operator a limit of the number of tries at any particular case, leaving an increased opportunity for other cases to be manipulated. This is an important feature as the Moreau operator requires a significant amount of computational resources for its frequent manipulation of the entire layout space.

The manipulations themselves are relatively straightforward, despite the large computational requirements associated with them. Yin and Cagan have mapped the performance of several pattern search techniques used for internal layout problems and found a performance difference of about 30% between the most basic coordinate pattern search and their best-performing, sophisticated conjugate gradient extended pattern search[138,xiii]. Despite the promise of performance increases offered by the more complicated techniques, the Moreau operator was based on the simpler technique to avoid complications with implementing a more complicated search in the midst of the GA already in place. Since the Moreau operator would not be acting on every case, the performance penalty would also be less significant.

[xiii] While the conjugate gradient extended pattern search did very well in Yin and Cagan's tests, two of their tested techniques, Rank Ordering Extended Pattern Search and Simplex Extended Pattern Search, did poorly, taking more than twice as long to complete the search as the default coordinate pattern search.

Initially, the plan was to move all components simultaneously, but that presented too many potential problems – how would the operator choose which direction to simultaneously move components? As the best movement plan for each component was highly dependent on all of the other components, a serial approach was finally reached. This was also the approach taken by Yin and Cagan in their searches, though their particular approach to layout-specific extensions was incorporated elsewhere as function evaluations would have to occur for cases that had not been modified by the Moreau operator[139].

The next step was to decide how much to perturb each component. Initially, a set of three moves of decreasing magnitude was to be performed on each component before moving on to the next one. Again, the serial nature of the problem made itself obvious – the later components would be too dependent on where the initial component had been moved. A compromise was finally reached that moved each of the components in order, then reduced the move limits and tried again. The single-direction search is common in path-building optimization and forms the basis for several of these methods, though this variant on a basic pattern search is among the most basic[140]. This was repeated as often as desired, though computation of the movement of each component became the single largest ongoing resource user of the optimization process. The normal range used for this work was between 3 and 7 iterations of the Moreau operator

The steps used to decrease the magnitude of the operations were initially set at a factor of two. This was initially determined to offer a fairly effective balance between aggressive movement and fine detail. After some initial experimentation, the Golden Section was used instead[141]. The slightly more

aggressive movement increased the convergence rate of several of the initial tests. This combined with its historic reputation to make for an effective step size.

In the end, the Moreau operator was designed with a good deal of flexibility, since informal experimentation found that the results were subject to variability based on tuning of the Moreau settings and the qualities of the particular problem. In addition to the percentage of each population group that would be subject to the Moreau operator, settings are available to set the number of move sets and the resolution of both the rotation and translation segments of the operation. This provides flexibility within the operator. It also allows the Moreau operator to subdivide the container space into finer increments than that used for the initial population generation. While a detailed exploration of this feature has not yet been performed, there is a possibility that the Moreau operator could be used to "nudge" components into positions not otherwise possible in arrangements that are making use of coarse subdivisions. Also, checks against rotation and translation limits for each component are performed within the Moreau operator, rather than in an external function as in most of the other code components. Future applications of a Moreau-style operator could also take advantage of the more advanced varieties of patterns searches available.

In tests using the Moreau operator with a demonstration code, the results were quite promising. Initial samples were run using the primary code with the volume calculation replaced with a simpler, less time-consuming process. The goal here was to minimize a set of simple computations. When the GA was run without the Moreau operator, the results were not particularly inspiring. Figure 27 shows the results of a 10-generation GA with 100 population members per

generation. As the generations progress, the results do improve, but only by a small amount. By the end of the test, a significant number of cases still have a value of more than 100, which is considered a poor result for this test. Further, none of the cases converged within 1% of the ideal.

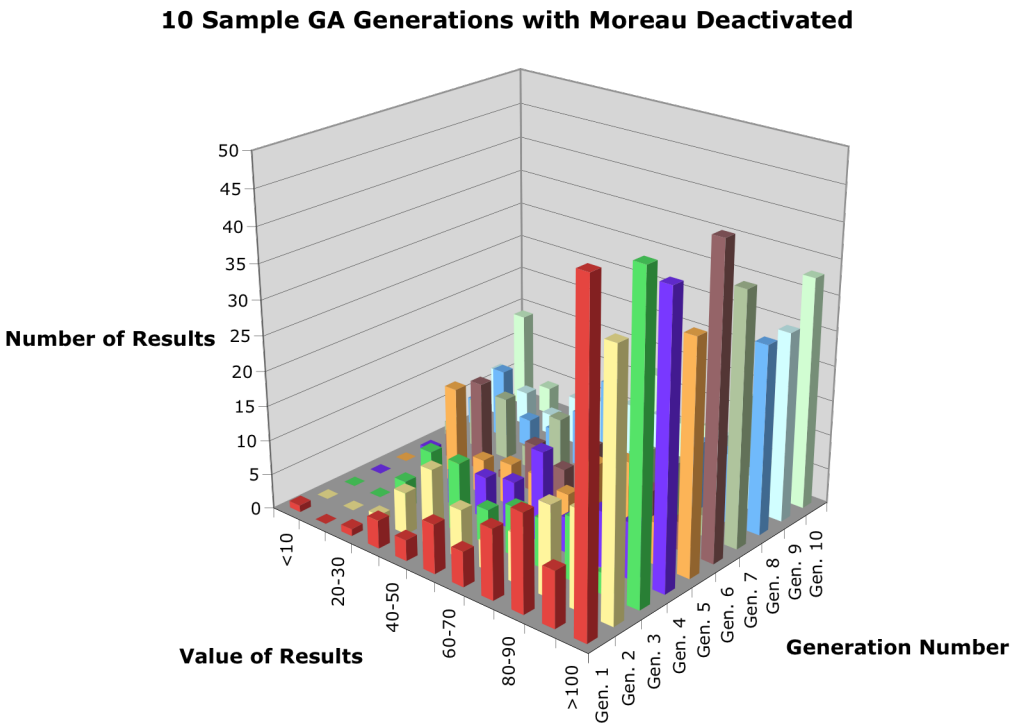


Figure 27: *Small test with no Moreau functionality*

Running the same test with the Moreau functionality turned on for all cases resulted in a significant change in the model's behavior. As shown in Figure 28, the Moreau had an almost immediate effect, dramatically improving results in the second generation. The red bars, representing the first generation,

are identical, but the remainder of the search space has shifted significantly toward the ideal solution. Results also showed convergence to within 1%.

The results were so dramatic that the author was concerned that the small population size was corrupting the results, so another set of tests was run on a population size containing 1,000 members per generation. The results were qualitatively identical, and are shown in Figure 29 and Figure 30. The only unexpected result in these tests was a surprisingly large number of cases in the "best" column for the case where the Moreau operator was turned off. This phenomenon was not present in the small test. Further investigation suggests that the number of "keepers" kept by the GA as part of its reproduction process in the larger test was capturing a significant number of points repeatedly at each generation, resulting in a large number of duplicate points. While unusual, this was a feature used in an attempt to promote mutations of the best cases. This phenomenon was also experienced in tests using the final code, as seen below, but the small size of the initial test prevented the characteristic from being revealed.

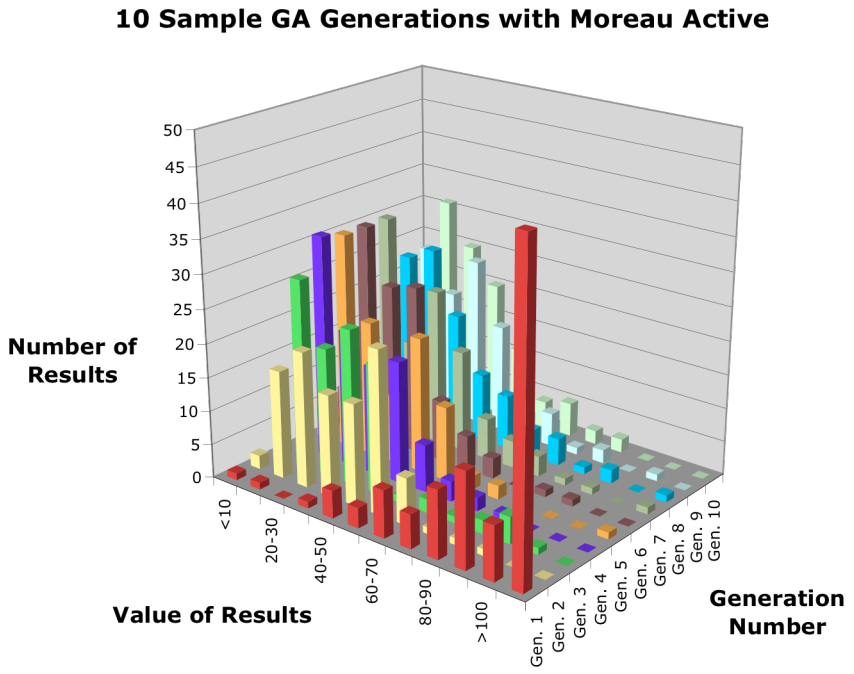


Figure 28: *Small test with Moreau activated*

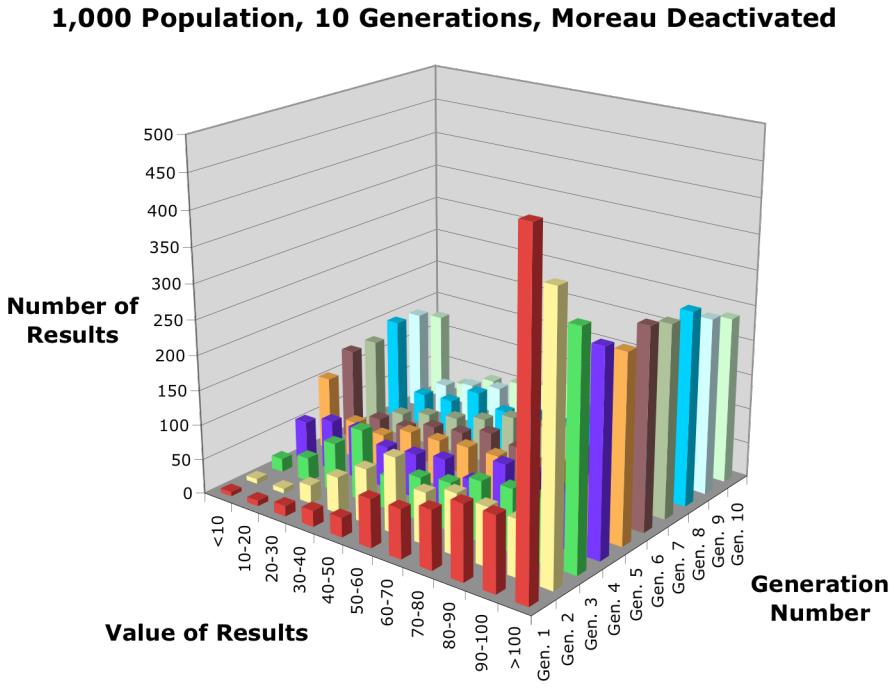


Figure 29: *Large test with Moreau deactivated*

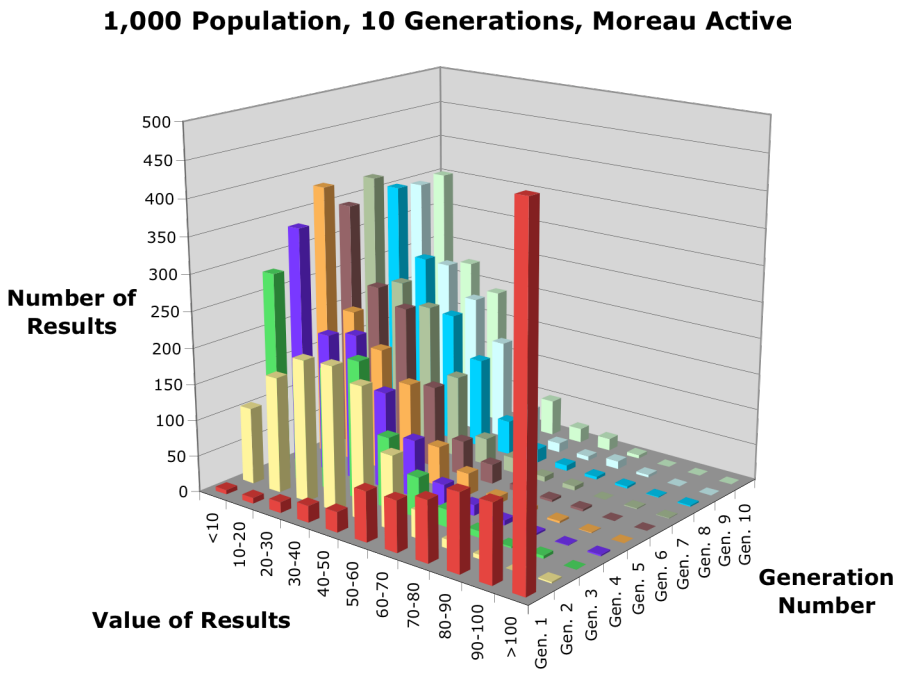


Figure 30: *Large test with Moreau active*

Based on these tests, the Moreau operator does have a dramatic effect on the convergence of the type of GA being used here, though some of its particular characteristics would doubtless benefit from tuning. The large drop in the first generation, followed by a much less obvious improvement in subsequent generations is intriguing, but it will be left to later tests in Chapter 7 to determine if similar behavior exists in the more sophisticated test problems.

6.3.5 Integration Software

System integration tools are commonly used by designers and other system-level analysts to organize a set of tasks and simplify the process of completing a complex task. These applications have demonstrated the ability to

produce significant time savings for a host of problems. These integrators, however, are not without their own problems. License and platform issues mean that the ability to run code on any computer at any time may be compromised. This is not to say that they could not be useful. In fact, a commercial application of volumetric sizing should make use of this type of tool, as it will help integrate the volumetric components with the remainder of the sizing process.

For the purposes of this work, commercial system integration tools from Phoenix Integration and Engineous Software were considered as standalone system integrators[142,143]. They were not used primarily for reasons of licensing and platform independence. They do offer a number of attractive features, however. In both cases the system integrators include access to a suite of optimizers. This would have made the comparison of optimization techniques on real problems more of a reality. A pair of screenshots from ModelCenter showing a simple model and some analytical results are shown in Figure 31.

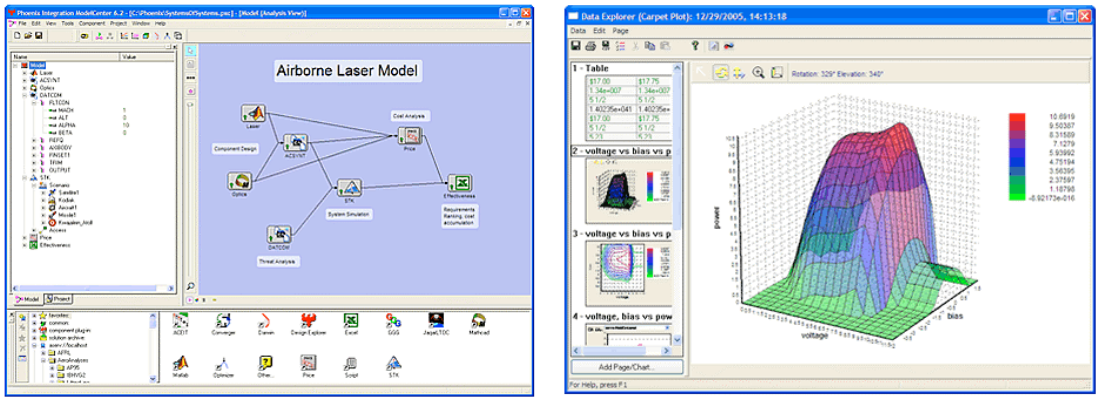


Figure 31: *Screenshots from ModelCenter*[144]

The C-based code framework being used is also considered to be more efficient than a commercial system integrator, as it relies on no complex software running at the same time as the basic algorithm; nor are there limits imposed on how the optimization scheme will operate, for example. Technosoft's *AMRaven* package, a front-end for their AML modeling language, offers many of the same capabilities needed for this work as the standalone system integration tools, but again, licensing and platform independence were among the issues that kept it from being chosen[145].

6.4 Problem Construction

In order to succeed, any toolset used for volumetric sizing must be able to perform the basic functions outlined in Section 4. These functions are briefly presented here with their representations within the tool:

Definition of container. The space that will hold the components

> The container is a GTS surface read from an external file. The surface needs to be relatively simple with a well defined inside and outside. GTS is capable of generating complex surfaces as well as those that are not physically possible, and these will not work reliably. A more detailed description of GTS' limitations can be found in Appendix B.2. GTS can also generate surfaces with detailed intrusions (such as ribs and stringers); while these surfaces should pose no problems for the toolset as it exists, these surfaces have not yet been tested. The surface should be centered at the origin by the user at its center of mass to help the code calculate center of gravity and other related information for the entire system. The code will then move the container (also called the fuselage within the code notation) to its center of volume to make better use of the placement

techniques used elsewhere in the code. Center of mass information is retained in the surface metadata that is generated within the code. GTS comes with utilities for converting standard .STL files into GTS files (and back again).

Definition of internal components. The components that will fill the container

The components are similarly GTS surfaces placed with their centers of mass at the origin. This simplifies the calculation of center of gravity and related details as above. All rotations are done about this center of mass. A small standalone piece of code was also created to center the components at the center of mass that GTS calculates based on the surfaces themselves. This only works for components that are homogeneous in nature (and as a consequence have a center of mass that coincides with the geometric center of the surface). Components also need to be relatively simple in the sense mentioned above. This does not, however, mean that components cannot be sophisticated, with multiple parts connected to each other, though these types of surfaces have also not been tested.

Placement of components. With all six degrees of freedom[xiv]

A short study of techniques for performing translation and rotation on an object in space demonstrated that the favored technique (that suggested by Mathworks, for example) is to move the object to the origin, perform the rotation, then move the object to the desired location[146]. This works well with the basic scheme GTS uses for rotations and translations. Using a single matrix to perform both features simultaneously in GTS results in a revolution of the object about the origin at a distance equal to that of the translation, producing rather different results than those desired. Leaving

[xiv] The words rotate and revolve both involve movement about an axis. For the purposes of this single paragraph, a revolution is considered the movement of the component about a point away from the component, much as the Earth revolves around the sun while rotating about its own axis on a daily basis. Confusingly, this motion is called a rotation in much of the supporting documentation used as a basis for this work, but the technique used to rotate the components should eliminate the chance for confusion elsewhere in this work. For the remainder of this document, rotation does refer to a component's movement about its own center.

the components at the origin for the rotations before they are moved for final placement removes this difficulty. Placement is then handled either by random numbers or by manipulations of one of the optimization tools. Rotation and placement limits are also handled through metadata files associated with each component. These are generated (with a set of default limits that allow unlimited placement and movement) for components that do not have limits associated with them.

Verification of components. Intersections with each other and the container

GTS has most of the tools built-in for handling intersections, but it does not handle inclusions without additional code. If a component is completely enclosed in another component, GTS will normally not notice, so additional code was added to account for this possibility. GTS is capable of calculating intersection volume as well as minimum distance between two surfaces. The notion of minimum distance, while not initially intuitive, is actually quite sensible, as two surfaces actually have many "distances" from each other. Each point on the surface has a particular distance from each point on the other distance. GTS can calculate all of these, but the minimum distance is clearly the most important for this purpose. Calculation of total system intersection volume is handled by a separate function that feeds back to the optimizers. GTS does have problems calculating intersections between two identical components located at exactly the same position. This will actually crash GTS (and whatever program is using it). An error-check built in to the code detects these possible problems before calculating intersection volumes.

Rearrangement of components. Move components repeatedly

Rearranging components is addressed through a random number generator that produces the required six values for each component's location. The random values are then modified by the optimizers when necessary in cooperation with the components' movement preferences as stored in the metadata. The original surfaces are not actually moved within the body of the main program. Instead, the original surfaces are

passed to individual functions for intersection detection and recordkeeping.

Optimization. Find and present the best solution

Optimization is performed at the global level, with a genetic algorithm (GA), as well as at the local level, with components of the Moreau Operator, described above in Section 6.3.4, that operates as a component within the GA.

Constraints. Customer-required and physical constraints

Sample constraints are calculated in a separate function. These constraints can be whatever the designer desires, but are limited to relatively simple examples here. The user-weighted value of the constraints is combined with the user-weighted intersection volume to produce the final value that is used by the optimizers.

The basic demonstration of these ideas requires a set of test cases. In order to simplify the process as much as possible, the layers of the project were developed in tune with advances in complexity in the problem itself. Initial studies involved single components and containers and then developed into more sophisticated problem sets. The first real test case, for example, was a set of boxes within a larger box. The software tool needed to correctly place twenty small, semi-rectangular boxes in a larger space. None of the boxes was perfectly rectangular, offering small "imperfections" to allow the possibility of unusual orientations while still offering significant chance of attaining a correct fit. Each small box had a volume of 13.99 units^3 while the larger container had a volume of just over 1011 units^3. This task was accomplished in several steps, increasing in difficulty until the software could place all the components in the box while

varying all six degrees of freedom. The details of each of these runs are presented in Chapter 7, but a brief description follows.

6.4.1 Initial Test Runs

The initial run involved placing the small boxes within the larger unit with only random translations and no rotations at all. This problem was designed to be as simple as possible, yet it also represents the way a human operator would probably solve the problem. For this particular problem, rotating the sample boxes would not be of any benefit to the solution as all the shapes involved were mostly rectangular and there was plenty of space available in the container.

The sample was run in two ways: as a set of randomly placed components in the container, repeated until completion; and as a modification of the "next fit" algorithm. For the "next fit" variant, random positions were chosen for each component, then compared to existing components and only accepted with no intersections. The initial cases were run with the components aligned with the fuselage container. This aided convergence time, as the optimizers did not have to address rotations. Using the same basic algorithm, another test was run with both translations and rotations. This significantly slowed the convergence, as boxes chaotically placed in a container are not nearly as efficient as nicely aligned rows.

6.4.2 Basic Optimization

For the next phase of the study, optimization was turned on. The genetic algorithm with Moreau operator was applied to the same set of components and

container used in the problem. The overall convergence rate was slowed significantly by the increased difficulty of computing each case. The Moreau operator in particular was a large resource user, but the basic intelligence of the component placement also used a significant portion of available resources.

As a further sign of development of the tool, constraints were also incorporated into this phase of the research, but the constraints selected were extremely simple and easy for the optimizer to satisfy in most cases and used few computing resources.

6.4.3 Second Test Runs

The second test case involved a more sophisticated set of shapes being placed into a more tightly constrained space. This test case was designed as a benchmark for the time required to solve a basic layout with no constraints. Constraints were then added to explore the additional time required for the optimizer to overcome the computational difficulty the constraints added to the problem.

6.4.4 Robustness

The next step involved the integration of uncertainty into the process. In earlier tests, the fit of the components in the notional fuselage space was guaranteed by the relatively sparse nature of the layout. In that situation there was no need for a probabilistic treatment of the layout as it produced a 100% chance of fitting. In order to truly demonstrate the effectiveness of a robust approach, a problem must be arranged that does not always work. The solution

should exist somewhere between a set of components that fit and one that does not fit within a fuselage container.

For this work, the same basic shapes were utilized, but reduced in size to aid computational efficiency. Fewer components were used as well in order to speed computations. A range of component sizes was then placed in the fuselage container in order to determine at what point the components would no longer fit. This experiment was performed first on a small subset of shapes and then again as part of the final test case, described below.

The shape changes were done by photographically scaling the components to reduce the complexity of this test. Conceptually, any aspect of a particular component's shape could be changed in size for the purposes of this section of the problem. Bookkeeping would then become challenging, as each change would need to be kept track of separately in order to properly predict the effects of individual changes. These difficulties would not add substantially to the problem, so the simpler method was used. GTS is also well-suited to photographic scaling, though the changes appropriate to future studies will differ somewhat due to differences in the way individual graphics environments treat components.

6.4.5 Final Test Case

The final test case involved a demonstration of the software and process on a sample aircraft design. The sample problem here consisted of a notional UAV cargo volume. The components to be placed were sensors, controls, and other payload items. While based on real aircraft, the arrangement does not represent any particular UAV. None of the components are the size of the real

items, and the container space has been generated uniquely for this problem. This has been done to provide as general and realistic a problem as is practical.

Performance figures have been generated using the *X-Plane* flight simulator. These performance figures are then entered into the *FLOPS* aircraft analysis code to produce mission capabilities for the notional aircraft.

Aspects of the aircraft were then be modified in accordance with the results of the component placement. If the placement results in excess room, can that space be traded for improved performance? If the aircraft has insufficient room for the components is it smarter to redesign the airplane or simply relax the cargo requirements? These types of questions led to a final set of design decisions.

6.5 Final Software Architecture

Volumetric sizing is only one component of a complete aircraft sizing process. Figure 17 is repeated here as Figure 32. For this work, a simplified suite of design tools was used as a framework to demonstrate the effectiveness of volumetric sizing on an aircraft design.

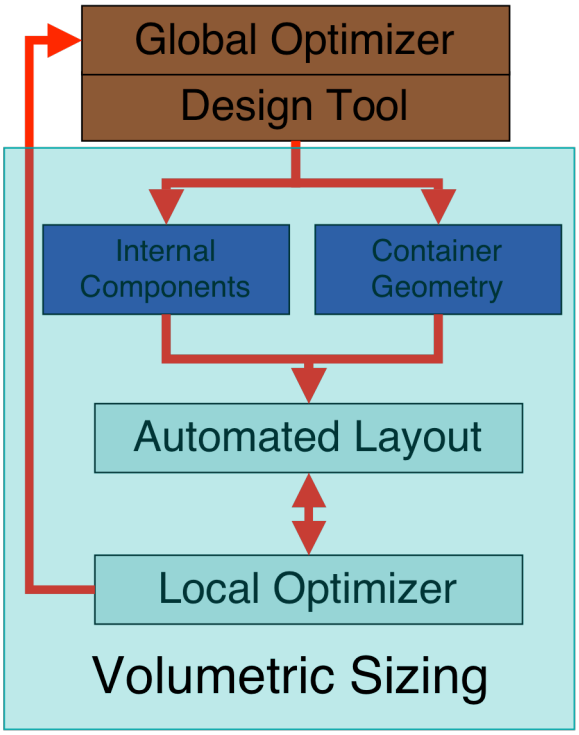


Figure 32: *Volumetric sizing as an addition to the design process*

While a more detailed exploration of the tools as used to complete the design in the sample problem is provided in Section 8, a basic outline is provided in Table 3. Component weights and aircraft structural needs were predicted from comparison to similarly sized aircraft, so no detailed structural analysis was performed. A functional decomposition of the core of the volumetric sizing process is presented in Figure 18, repeated again here as Figure 33 for clarity. A brief description of the functionality of each software component as relates to the flowchart is presented in Table 4.

Table 3: *Analysis software toolset*

Exterior Geometry and Performance:	*X-Plane*
Interior Shape Creation:	*Blender*
Mission Analysis:	*FLOPS*
Volumetric Evaluation:	Original GTS/C Code
Formatting/Intermediate Calculations:	*Excel*

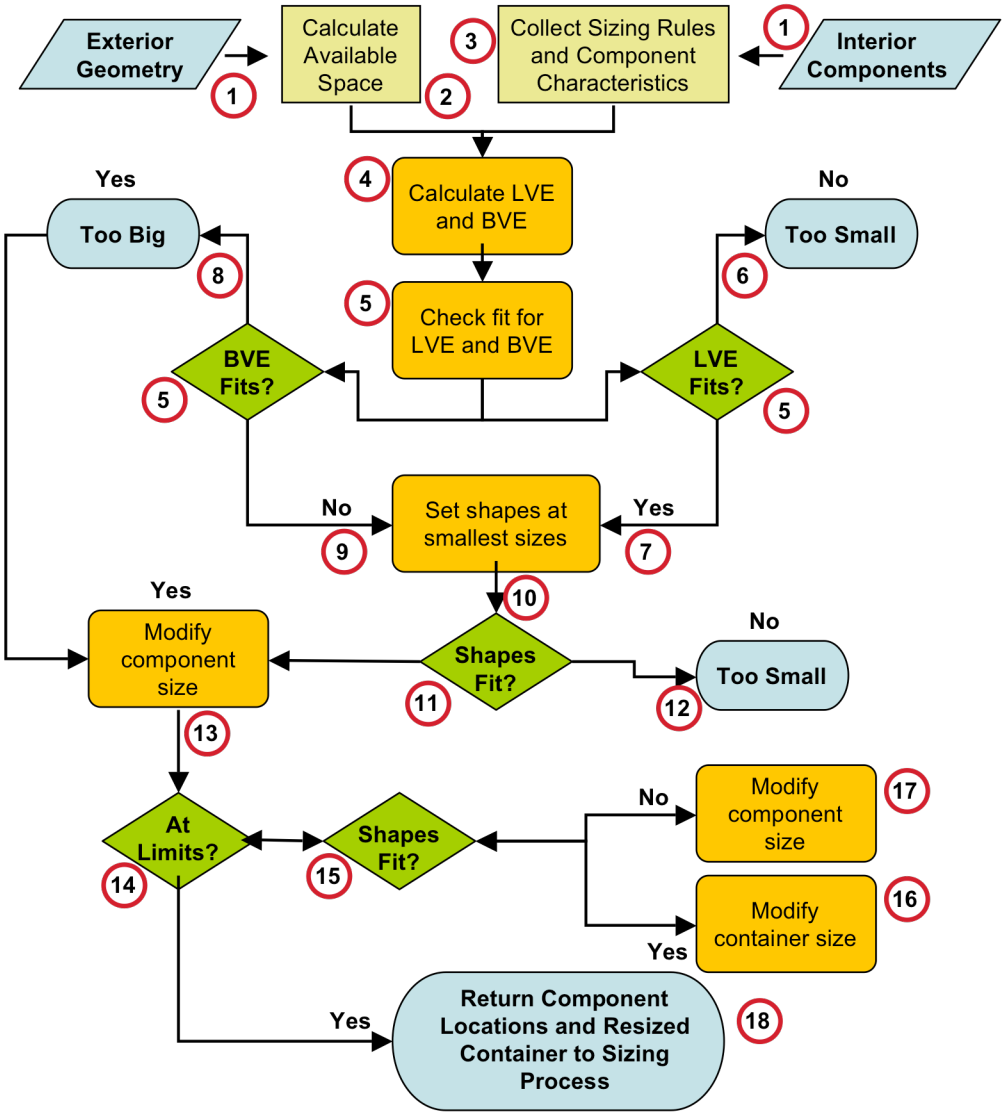


Figure 33: *Flowchart of volumetric elements in a sizing process*

Software components of the code developed for this tool are described in detail in Appendix A2, and existing software applications used in support of the original code are covered in more detail in Appendix B2.

Table 4: *Localized flowchart explanation*

1	All shapes created with *Blender*. GTS utilities used to convert to GTS format. Internal components from estimated specs. Fuselage from *X-Plane*-generated outer mold lines.
2	All shapes read in from external files. Available space calculation within the main program block.
3	Component characteristics, move limits, and other features read in from metadata files. If none exist, the *make_default* function generates metadata with default values. Constraints are currently hard-coded but could be passed as metadata.
4	Initial Volume calculations done in main program block.
5	Check violations are currently warnings in text output from the code.
6	If container is too small code sends text warning to output.
7 **8** **9** **10** **11**	Currently, the code only handles one size component. Also, GTS suffers little performance degradation with relatively complex surfaces, so BVE checks were not made in the experiments here. Changes in container and component sizes are made by hand, but provision has been included in the metadata format for automated scaling in future work.
12	Containers that are too small will result in a positive intersection/constraint result. The magnitude of the violation will suggest the magnitude of the needed correction.
13	Component sizes can be rapidly changed with the GTS *transform* utility.
14	Component size limits can be carried within the metadata but are currently recorded by hand.
15	Fit is again handled through the main portion of the code, with intersection checks and constraint violations driving the primary results.
16 **17**	Resizing of the container or components is again done through the transform utility.
18	Output from this process consists of component positions for the best arrangement found in the analysis process as well as associated intersection and constraint violations, if applicable.

The code itself created for this work is focused primarily on the portions of the process involving the fit checks themselves. Currently, most other functions are performed by hand or through the use of supporting applications, as described in Appendix B2. The basic code structure, however, can be explored simply in a table, as below in Table 5.

Table 5: *Code outline*

1	Main program block initializes all variables and user settings.
2	Main program reads in component and container shapes as well as metadata from external files. If metadata is not available, the *make_default* function generates default metadata.
3	With a complete set of surfaces now in memory, basic limits are assessed. *box_diagonal* sets initial limits and *all_box_volume* does initial volume checks.
4	The code now begins creating the first generation of position information. *make_random_point* is used to generate the points while *make_pop_member* makes sure they are valid, and the code checks them for fit if the intelligent placement is active.
5	*inside*, *check_intersect*, and *are_different* are used repeatably by the volume_check and main code block to check for inclusion, measure intersection volume, and check for identical component locations to avoid crashing GTS. *check_points* and *get_a_point* also support these functions.
6	With a first generation complete, it is now evaluated using the *volume_check* and *constraint_check* functions.
7	The reproduction phase of the GA is first, with the best results of the process being sorted into a list of "keepers".
8	*mutate_GA* and *crossover_GA* now operate on appropriate portions of the population, picking members at random and checking he results with the ok_changes function against limits from the metadata.
9	The *small_swap* function is used by both the mutation and crossover operators to actually change coordinate values.
10	The *Moreau_GA* operator now modifies selected population members. The Moreau operator first modifies members of the "keepers". If the number of Moreau operations is more than the number of "keepers", random population members are chosen for the "treatment".
11	Remaining population positions are now randomly filled with new members using the same technique as above and the "keepers" are re-ordered.
12	The basic process repeats until the run limit of the GA is reached. The process will end early if convergence to 0 violations occurs.
13	At the end of the run, the best remaining "keepers" are output, as are the gross results and histogram data.
14	Finally, the original and relocated components are output to disk for comparison and visualization.

The code is then applied in a sizing methodology like the one in Figure 8, again repeated here as Figure 34.

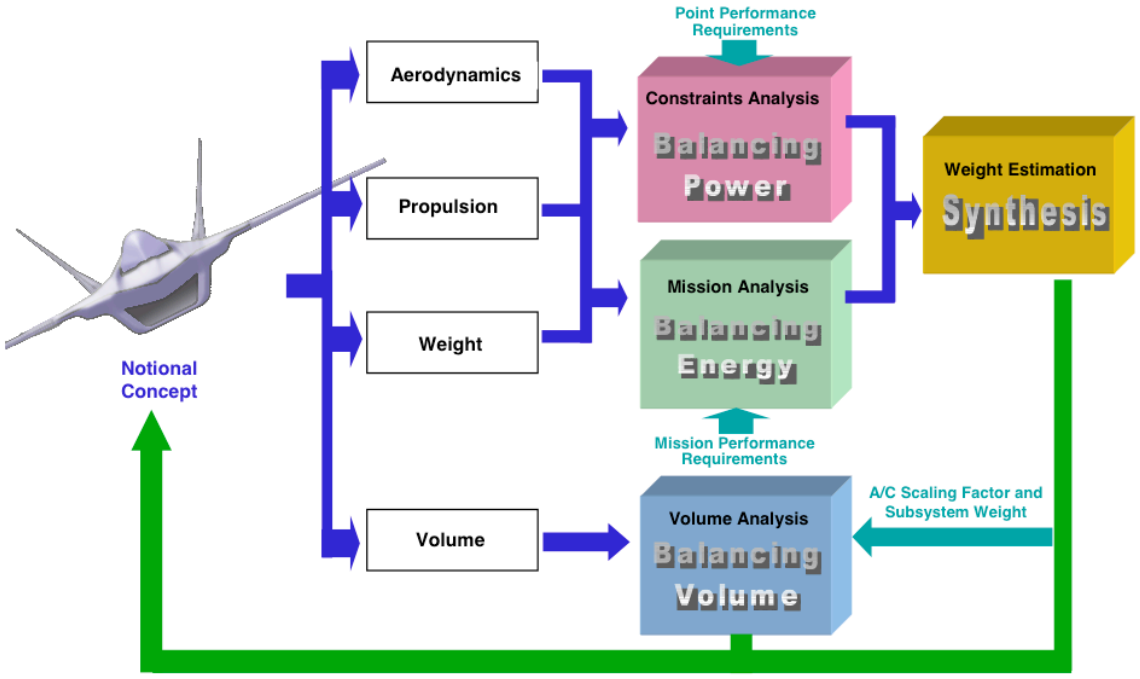


Figure 34: *A comprehensive aircraft sizing method (after Nam)*[147]

7 Test Case Results

The demonstration of the effectiveness of volumetric sizing truly begins with a set of test problems. Each more sophisticated than the last, the test problems were designed to add complexity to the problem as it developed. The final test case is the demonstration of volumetric sizing techniques on a sample aircraft design. That final demonstration is presented in the next section. The test cases presented here represent only the later portions of preliminary testing. Numerous tests were run with limited component groups and "dummy" calculations that only served as basic proof-of-concept tools for the primary pieces of the final code.

Explorations around the base test arrangement are also presented here. Perturbations in the arrangement of the Moreau operator, the characteristics of the genetic algorithm, and the number of runs/cases were all modified during the course of testing. This produced some significant changes in results, which will be explained below.

7.1 Initial Case

The initial test case involved a relatively simple fuselage shape with twenty similarly-shaped (though smaller) components to fit inside. Volume of the "fuselage" was 1011units^{3}[xv]. Volume of the components was 13.99units^{3} each.

[xv] No physical units were associated with the parts at this point, as they do not exist outside of the modeling environment. GTS uses raw values with no

The initial runs were made with no constraints at all and only translations (no rotations). Two versions of the code were developed in parallel for this portion of the work. The first was designed to emulate the quantities of run numbers that would be needed when the domain-spanning optimizer was activated. The second version of the code was designed to complete a design in a single iteration, using an extremely basic form of machine intelligence related to the next-fit algorithm described earlier.

7.1.1 First Results

Initial component arrangements within the code were handled randomly. The fuselage surface was moved so its center coincided with the origin. This made the system symmetric and reduced the amount of "extra" space outside of the fuselage that the code would have to search. Since only simple forms were being considered at this point in the process, the center of mass of the object was considered to be the same as the geometric center. For the purposes of later work involving center of gravity calculations with more realistic components, this would not work, but for the simple purposes of placement, it was relatively efficient. Later work kept the actual fuselage center of mass in a separate piece of metadata that accompanied the surface within the code.

A bounding box was then created around the fuselage space in order to help determine the maximum distance that would be appropriate for components to be moved while still restricting them enough to fall inside the fuselage with a high amount of regularity. The bounding box was aligned with

---

associated units, so elements of the study can be scaled arbitrarily to match a desired unit basis. In this case "unit" is a measure of length, with unit^3 being a volume value.

the three axes and completely enclosed the fuselage container. The length of a semi-diagonal of the bounding box created a sphere around the origin that contained every point of the fuselage. Not centering the fuselage at this point would have resulted in a sphere that contained more "extra" space than necessary. A two-dimensional representation of this scheme is shown in Figure 35.

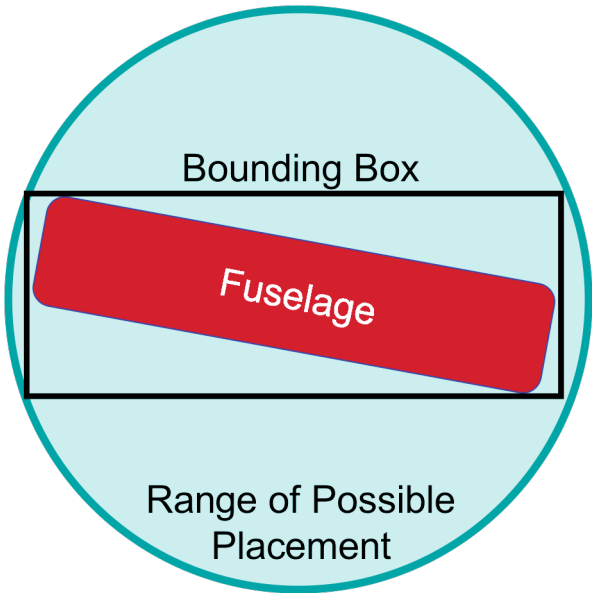


Figure 35: *Two-dimensional placement example*

Components could then simply be translated through a random distance in each axis to a maximum of the bounding box semi-diagonal. While this process did guarantee that components would not be placed a great distance away from the fuselage, there did remain, even in the simplified test case, a significant amount of space that components could occupy that was not in the fuselage. While this placement inefficiency should not be a problem for any

domain-spanning optimization, as a purely random exercise, the results were poor. In a test run with the simplified test fuselage and 20 simple components, using rotations and translations, a run of over 160,000 iterations produced no solutions, and few cases that were even close, as explored below.

Solutions at this point in the code development were evaluated simply on the basis of intersection volume. GTS calculates component intersection volumes directly for component/component intersections. For component/fuselage intersections, the GTS results were modified. Since component inclusion within the fuselage is the desired result, components are considered to intersect the fuselage if they fall outside it. Similarly, the portions of intersecting components that fall outside the fuselage, rather than the parts that fall inside, are counted toward fuselage/component intersections. This calculation is simple, requiring only an additional component volume calculation and simple subtraction in order to create the correct results.

The best result from the run of 160,000 was an intersection volume of over 133 units^3, which is not much better than the approximately 280 units^3 of total component volume. A significant contributor to these results was the large amount of volume included in the sphere that was not contained in the fuselage space. The fuselage volume, as calculated by GTS, was 1011.74 units^3. Using the bounding box semi-length of 8.869 units to create a sphere results in a sphere volume of about 2922.83 units^3 of volume. This means code using only this simple distance limit for placing components will place close to two in every three components outside the fuselage. With a more conventionally shaped fuselage, poor placement would be even more likely.

The first step in eliminating this large surplus of useless cases was to restrict the component placement to cases that either kept components entirely within the fuselage or at least touching the fuselage. This would allow "close" solutions to be considered by later optimizers while potentially improving the chances of acceptable solutions with just the random placement. In test runs of 100,000 cases, this strategy did not improve the results as much as anticipated. Several cases did come close to solution, but the resultant overlaps (between the components and the fuselage and within the components themselves) were still on the order of two entire components.

Even worse, runtimes on the test computers were measured in days, with the Macintosh system requiring just over two days of user time and the Linux system around four days. This was almost an order of magnitude slower than the initial random runs. Some of this time penalty was later found to be due to memory leaks within the code. Even when the code was modified to remove most of the memory leaks, the results were similar, with the Linux system still taking more than two days. Another issue was that initial runs were revolving the components around the origin rather than rotating them around their own centers of mass. The solution to this issue is explained below in Section 7.1.3. With those issues fixed, the efficiency of the code was greatly improved, and while runtimes did not improve significantly, the quality of the results did.

The next step involved the elimination of all fuselage overlaps, not just the cases with components completely outside the fuselage. This would not necessarily be desirable for processes that involved complex optimization, since some undesirable points could help the optimizer find solutions with components near the fuselage boundary, but for cases using random placement,

the ability to keep all the components completely within the fuselage space improved the quality of the solution significantly.

While not much slower per iteration than the first simple restriction (around a factor of two), this version did result in generally smaller intersection volumes for all cases. The initial case, for example, resulted in an intersection volume of 74.12 units^3 – better than the best of 160,000 cases in the initial test. In 60,000 cases on the Macintosh system, the best result was an intersection volume of 13.68 units^3 in a little more than 39 hours of user time. A second test using a slightly improved version of the code, changed for consistency with other components being developed and with fewer memory leaks, resulted in similar answers, with a best point of 20.51 units^3 being found in less than 12 hours.

7.1.2 Eliminating Intersections

Eliminating fuselage and component overlaps at the same time promised a much quicker solution. Indeed, using this type of solution would seem to be the most appropriate answer to placement in general, and the computer intelligence required to implement it would need little in terms of resources. Unfortunately, in cases where no valid solution exists, any attempt to completely eliminate intersections would result in an infinite loop. The addition of constraints to the solution space may also preclude this type of solution, as constraints are difficult to assess until the end of the run, as explained above. In an attempt to explore the possibilities of solving the layout problem in this manner, however, a separate version of the code was developed.

Since every intersection should be eliminated, the need for multiple cases was removed. With a single run, the code promised to be much more rapid than

the thousands of cases run in the normal code, even with the additional comparisons involved. A catch variable was added to the code to prevent infinite loops.

The initial results from the code were impressive. For the first set of results, no rotations were allowed. While initial runs bumped into the catch variable and resulted in component positions that still produced intersections, either with the fuselage or each other, expanding the catch variable allowed a converged solution with no intersection in less than a minute of user time. The lack of intersections was verified by placing each of the moved components in the *Blender* visualization software with the fuselage[148]. Indeed, the components were placed properly, with no component/component or component/fuselage intersections as can be seen in Figure 36. The outer line of the fuselage container is represented by a black wireframe, while the interior components are shaded grey boxes. The grid, pyramid, axis marker and reticle are all artifacts from the *Blender* visualization software.

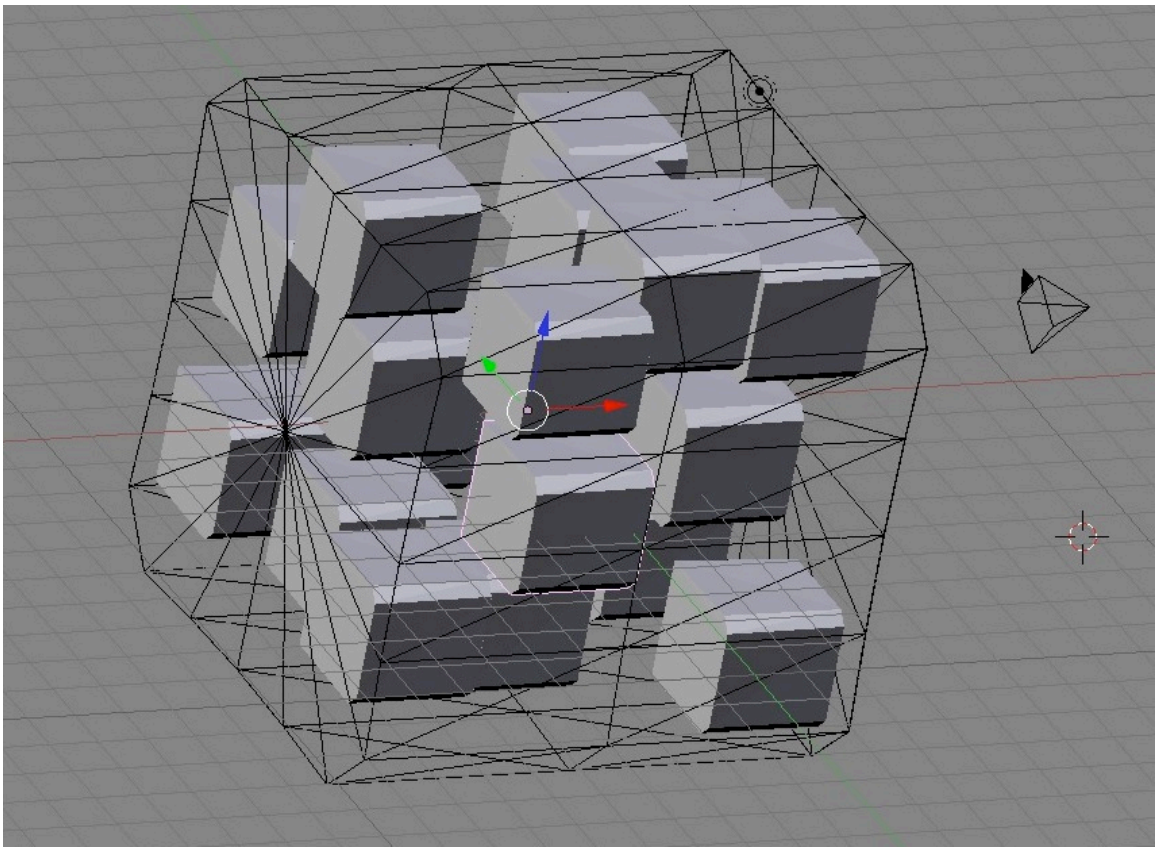


Figure 36: *20 Simplified components in the test container*

7.1.3 Adding Rotations

Adding rotations to the components increased the time needed for analysis. In addition to the simple combinatorial increase, rotations added a greater level of difficulty since the easiest solutions in this case required no rotation at all. The wrong set of rotations, in fact, could cause an impossible arrangement, even in a relatively sparse container such as the one used in this test. Initial demonstrations of this arrangement were completely unsatisfactory, as the computer would run for nearly 24 hours without reaching a solution. Even

with relatively high settings for the catch variable, the code would not converge before it crashed because of the memory leaks mentioned earlier.

The code also suffered from a poor choice of rotation schemes as mentioned above. Initially, the revolution of the components was handled as part of the same matrix as the translations. This produced components that did not end up where they were expected to go. If a component was to be centered at {1,0,0} and then rotated 180 degrees in the x-axis, instead of being turned around in place at {1,0,0}, it ended up at {-1,0,0} after having been revolved 180 degrees around the origin. For this particular problem, this made the convergence process even slower. The solution, as mentioned earlier, was to leave the component at the origin, rotate it there, then move it out to the desired location.

With the memory leaks and rotation scheme repaired, the same exercise was run again, with a converged solution occurring in less than six hours on a 2Ghz iMac G5. This solution can be seen below in Figure 37.

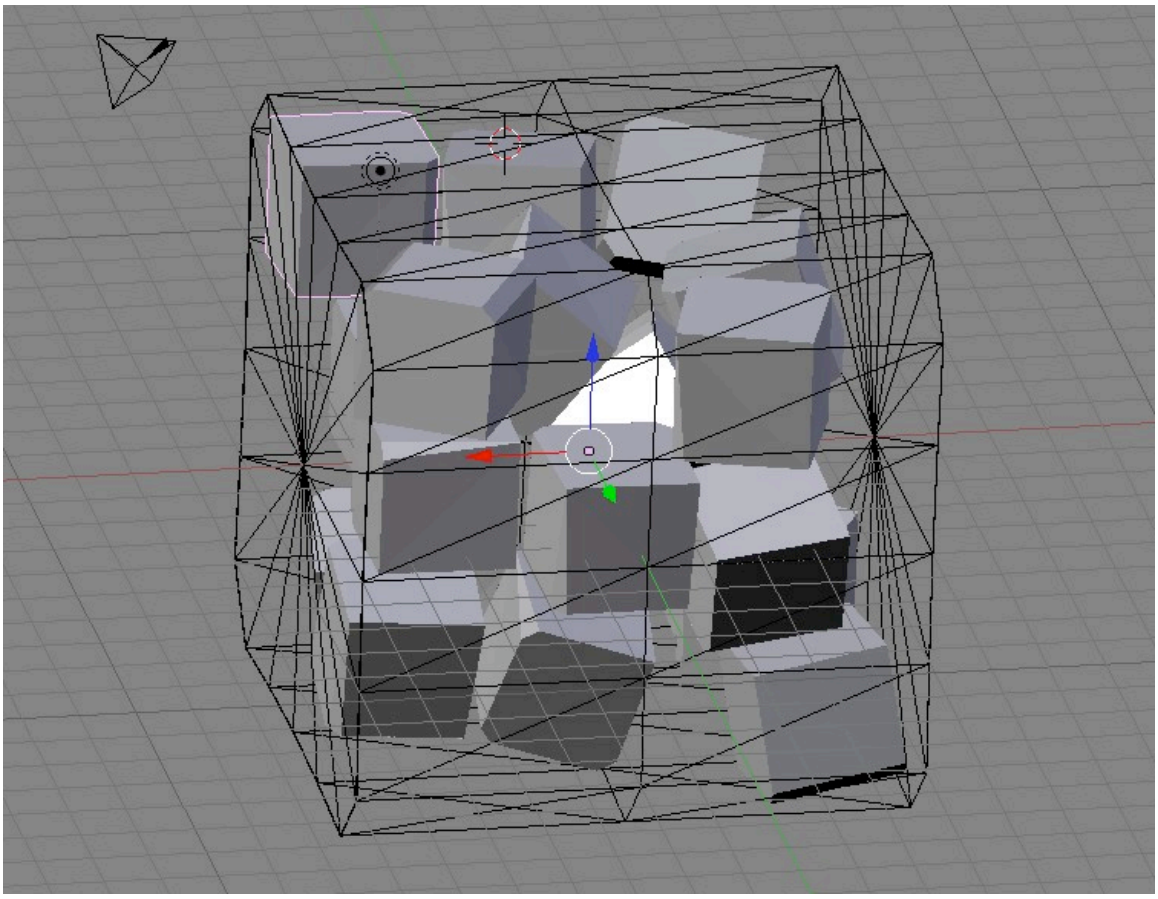


Figure 37: *Simplified components with rotations in test container*

This section of code development also marked the first attempt to dynamically adjust the resolution of the component layout. While all initial attempts assumed 65,535 divisions for each axis of translation and 180 for each axis of rotation, functionality was added to allow scaling of this resolution by the user. In initial experimentation, shifting the resolution did nothing to change the results or the time required to attain them, though extremely coarse resolution settings did highlight several unrelated underlying issues through a more rapid exploration of more portions of the design space. More work needs to be done in

this area, however, to fully ascertain the effectiveness of changing the search resolution.

At first glance, using a next-fit arrangement such as the one just shown would seem to be an effective technique for arranging components. As will be seen in the sections below, a six-hour runtime is quite short for a problem of this scope when compared to the techniques used later. The problem is not, unfortunately, that simple. Difficulties arising from ordering and constraints can cause this technique to fail to produce a solution even if one is possible. A solution to this problem would be to simply extend this technique with a domain spanning optimizer and run repeated cases over the entire design space. While that may be possible at some point in the future, or for simple cases, the computational requirements to perform a six-hour optimization thousands of times exceed the practical limits of an effective volumetric sizing tool.

7.2 Optimization

The next step was to implement the optimization tools. This required another fundamental rewrite of the code structure. While most of the functions from earlier versions of the code were retained relatively intact, the new version of the code was structurally different in order to account for the optimization tools. Additional functions were added to implement the genetic algorithm and its associated Moreau operator. Other functions were also added for error checking of duplicate component locations and other small items. Flexibility was also enhanced, as all aspects of the code were now able to adjust their resolution

dynamically based on user input. Details of code development are available in Appendix A.

The optimizer allows user control over the population size, number of generations, and percentages of each of the GA operators. While currently activated as constants in the source code, a future code could make use of these features as program arguments or information read from the input files. A table of optimization variables available is shown in Table 6.

Table 6: *User-Controlled Optimization Variables*

Variable	Value	Default	Notes
MAX_RES	integer	65535	Maximum number of divisions allowed in a space
RES_MULT	integer	500	Multiplier used with dynamic resolution option
DYNAMIC_RES	boolean	FALSE	Set to TRUE, allows resolution to set itself
IS_SMART	integer	1	Set to 0, allows components outside the fuselage
GA_DFLT_POPSIZE	integer	2000	Number of population members in the GA
GA_DFLT_RUNLIM	integer	20	Number of generations for the GA
GA_XOVER_PERC	real < 100	40	Percent of each population that will have crossover performed
GA_MOREAU_PERC	real < 100	10	Percent of each population that will have Moreau performed
GA_KEEP_COUNT	real < 100	2	Percent of each population that is kept as the "best"
GA_VOL_WT	real	1	Weighting for the Volume score
GA_CON_WT	real	1	Weighting for the Constraint score

7.2.1 Initial Results From Optimization

The first step involved evaluating the basic techniques for speed and efficiency. Using a purely random placement technique with no guarantees of placement within the fuselage produced extremely fast results, allowing better than 1,000 cases per hour on a 2.0Ghz iMac G5. The same simplicity that generated such rapid cases, however, produced results in the test cases that never approached a solution. In each of these cases, the components were capable of fitting into the fuselage with no intersections at all. These test cases used the same 20 components, each with a volume of just under 14units^3, yet the final results from these initial runs, even after 160,000 cases, still produced over 100units^3 of intersection. With an optimal value of zero, a result that high was not encouraging. Figure 38 shows a comparison of the different optimization schemes used at this early phase of the work.

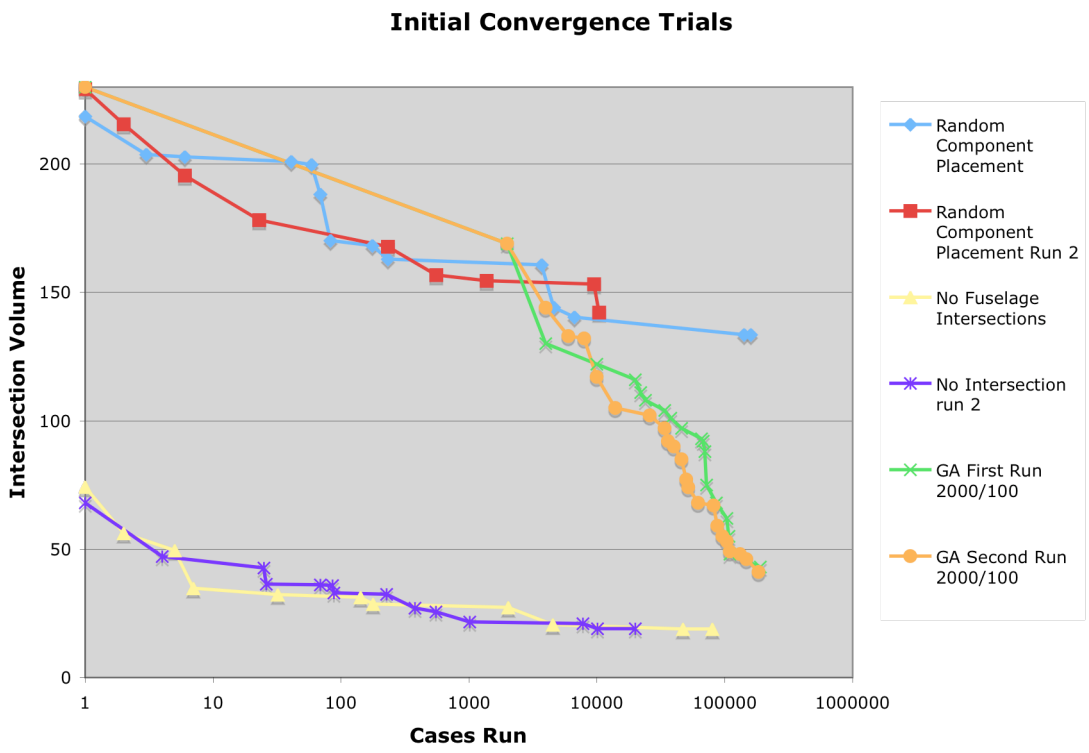


Figure 38: *Initial optimization trial cases*

The random component placement cases represent the cases that lacked even basic placement intelligence. On average, these cases tended to put close to two out of every three components outside the fuselage, yet still in the allowable area. This produced the poor results mentioned above. As can be seen in Figure 39, this placement technique generates a nice, smooth histogram that, regrettably, suggests that convergence to the desired zero intersection volume will almost never happen.

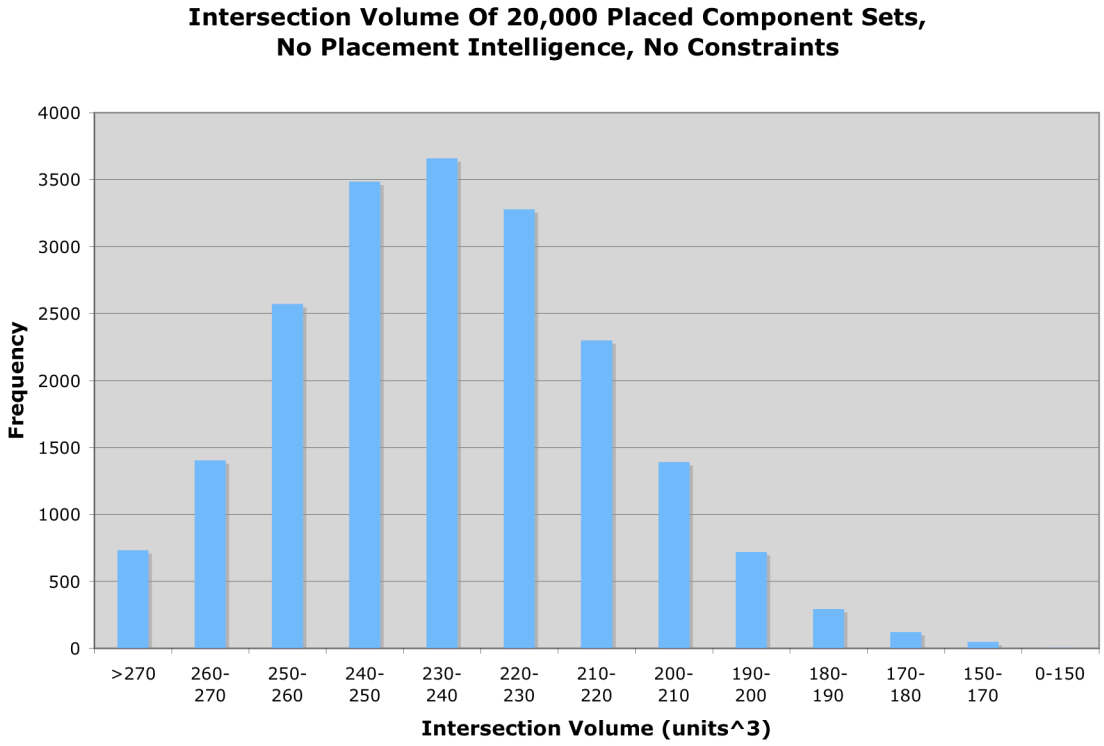


Figure 39: *Histogram of random placement intersections*

The first step in improving the process was to add at least enough intelligence to the process of component placement to get each component into the fuselage container. This was accomplished by adding two steps to each component placement. After the initial random coordinates were created, a check was first made to see if a point on the component was inside the fuselage. If the component object passed that test, an intersection check was also made. Failing either check meant the component was not completely inside the fuselage and needed a new set of coordinates. This process was repeated until the component was completely inside the fuselage. Since the basic process made such poor

initial placements, this additional computational load significantly extended the time required by each case.

Fortunately, that time was not wasted. Using this guarantee of fuselage inclusion dramatically decreased the initial intersection volumes for nearly every case run. This produced a far better starting point from which to attempt to converge to zero. The initial cases using the more intelligent placement technique had less intersection volume, in fact, than the best cases ever attained using the initial method. Despite the time penalty incurred by this system, results improved from there, with best cases usually having less than 20units^3 of intersection volume after between 20,000 and 80,000 cases.

This also improved the results in the histogram. Figure 40 shows the results from the improved method. Interestingly, the shape of the histogram is not nearly as smooth as the histogram using purely random placement. This may be due to the constraint of having to fit the components into the fuselage space. A significant number of possible component arrangements are therefore eliminated, leaving a grouping that has to conform to the shape of the fuselage.

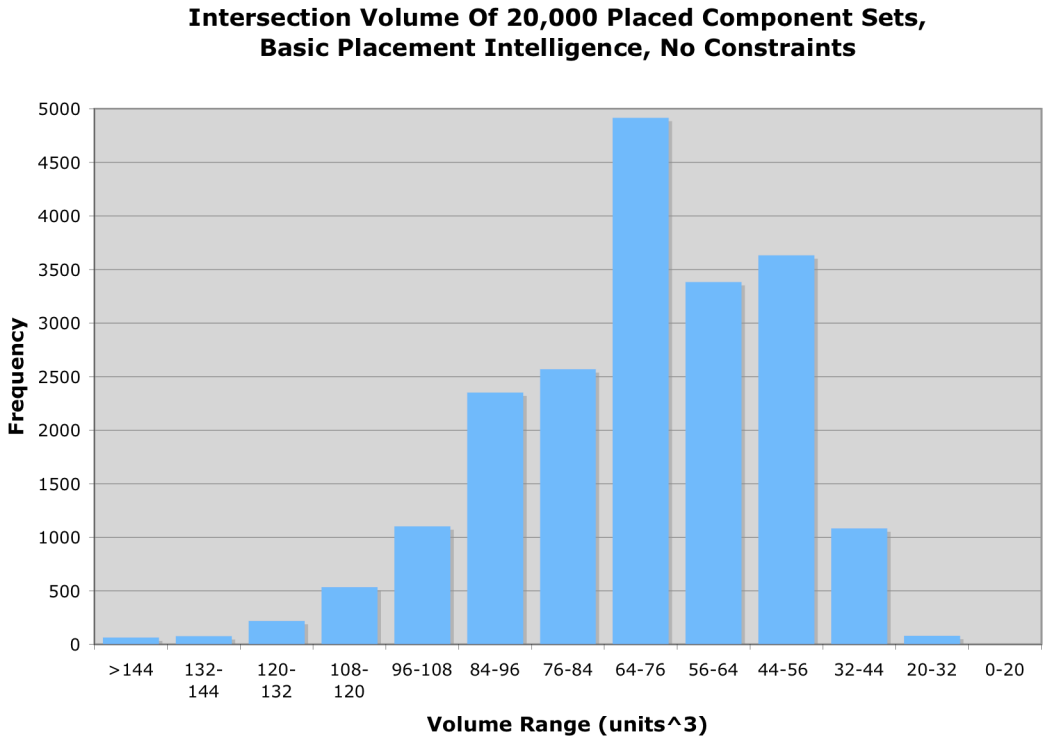


Figure 40: *Histogram of intelligent placement intersections*

The next step was to activate the genetic algorithm (GA). In the first several runs, the GA was operated over the basic placement technique. This allowed the GA to run rapidly over a large range of cases to help debug potential problems with the GA itself. The results were encouraging when compared to the initial random runs. Even without the Moreau operator, the GA still produced a significantly better convergence rate than the basic random cases. In the cases shown in Figure 38, above, the GA converged to a value of about 43units^3. While this was not as good as the "intelligent" placement cases, it was far better than the random cases had been.

It was at this point that an interesting aspect of the GA was discovered. Genetic algorithms, like most domain-spanning optimization techniques, operate best after tuning to make them better suited to the problem at hand. One of the reasons a GA was selected for this particular work was the generally high level of robustness that GAs possess. While the GA used here makes use of the traditional operators, reproduction, mutation, and crossover, the crossover operator was found to be almost useless for most of the problems using this group of sample components, since they are all identical. When the coordinates for different components were exchanged during the crossover operation, there was no effect. This was found to be caused by a particular side-effect of having identical components in the sample problem – swapping the coordinates of identical components produces identical results. This should not be a concern in tasks using a wider variety of components, but for this problem, it meant that the crossover percentage for each generation of the GA was lowered to nearly zero in future tests. The small number of crossovers preserved in the process ensured that the crossover function was at least functioning properly. Some of the crossover cases were replaced with "new blood", random cases placed in the population as the original cases had been.

The GA was generally effective at reducing the overall value of the intersection volume even with the Moreau operator turned off. A small sample run is displayed in Figure 41. A few features of this example help characterize many of the qualities of runs made with this optimizer. One feature that will strike many users as odd is the jump in poor values (over 100) after the first generation. This is a result of the way the code places components. During the initial population generation, all components were required to be completely

inside the fuselage container. For subsequent mutations and crossovers, the restriction was not enforced; some of the subsequent population members will have intersection volumes higher than those in the initial population. This feature was allowed to remain in the optimizer since it could contribute to the optimizer's potential discovery of good solutions near poor solutions. New random cases placed as part of the GA did meet the fuselage inclusion restriction.

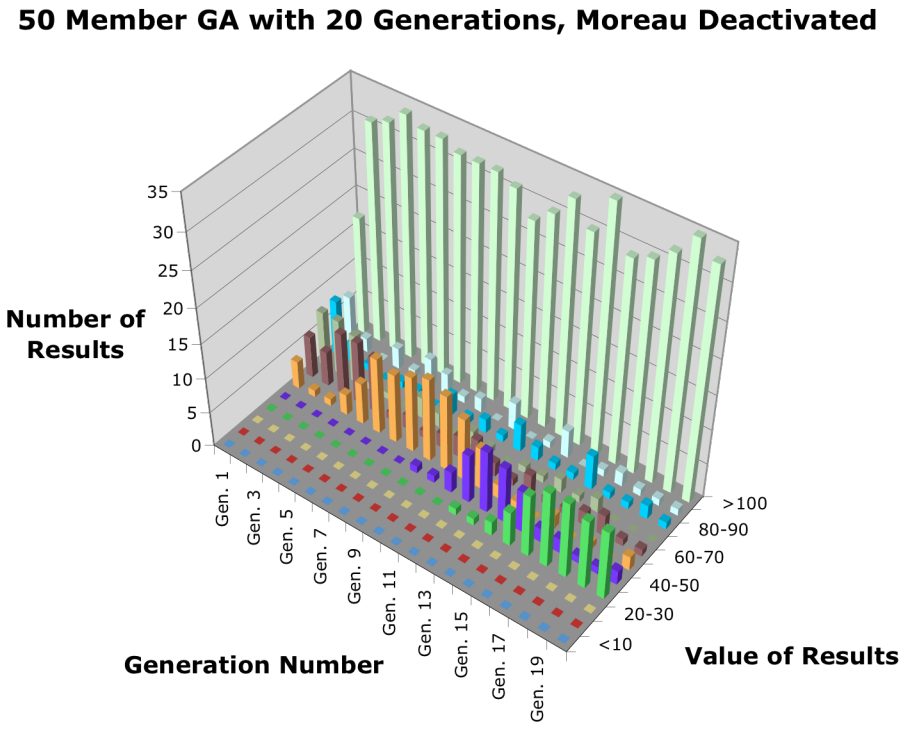


Figure 41: *Small GA with Moreau deactivated*

Another apparently unusual feature is the lack of any solutions close to zero, the optimum. With so few cases run, and without the advantage of the Moreau operator, the results here, while better than the random cases, are still

not optimal. One reason for this was thought to be that the results presented in Figure 41 were done with basic constraints turned on. Further study showed that this actually had only a small effect, as few of the best population members violated any of the included CG constraints. Those that did were only present in earlier generations and were removed by the end of the process. As mentioned earlier, extending the population size would result in improved answers, but without help from the Moreau operator, none of the results were better than around a volume of 13 units^3.

Activating the Moreau operator had a significant effect on the convergence, both in terms of rate and quality. While the deactivated case in Figure 41 did not produce any values less than 20 units^3, using the Moreau operator in an otherwise identical run resulted in convergence to essentially zero units^3 of intersection volume. A value of zero, representing no intersections, is considered the ideal. The results from this test are shown in Figure 42.

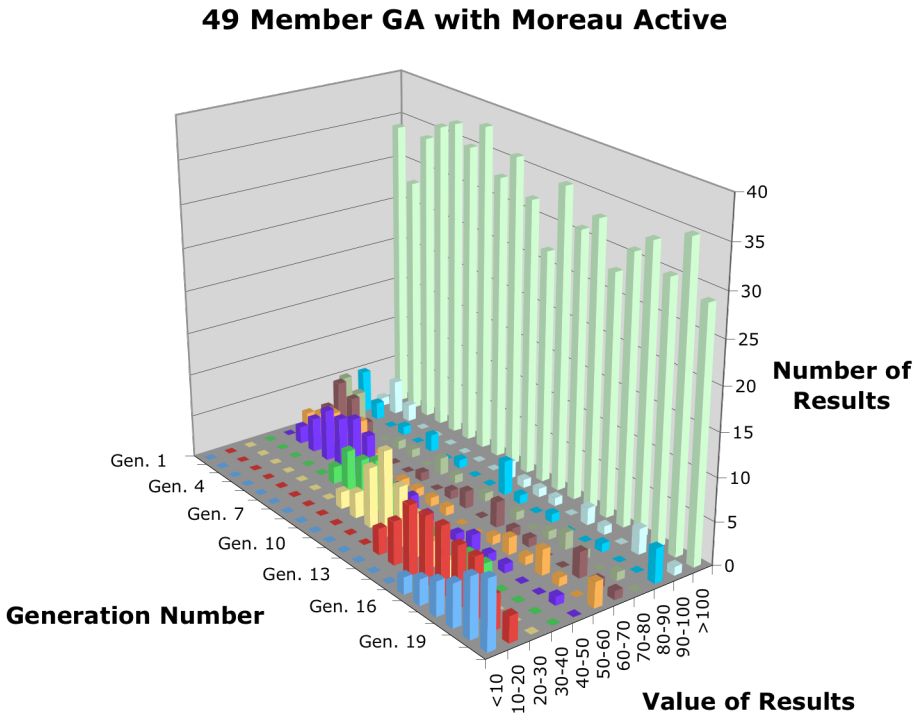


Figure 42: *Small GA results with Moreau active*

A quick comparison between the results shown in the two tests illustrates several points. The Moreau case converged much more rapidly, beating the deactivated case's best result at the 14th generation out of 20. The Moreau case also converged to a better overall solution, with two cases that had less than one unit^3 of intersection volume. This also occurred while meeting the simple center of gravity constraint (which was not active in the earlier case).

A more detailed investigation shows that the two techniques do share a similar convergence pattern. With wave-like patterns following the progression of "best" results, each technique featured a small subset of better results overshadowed in quantity by a large number of relatively poor results. As

mentioned earlier, this result is due to the dramatic effect of the random elements within the GA, allowing component arrangements that frequently intersect the fuselage and each other. These poor arrangements are not entirely wasted. A more detailed investigation of the result data indicates that mutation and crossover of some of these elements can result in relatively good results, particularly in earlier generations of the GA. Several of the spikes in the middle of the result values in both figures are due to this phenomenon.

But how does this affect overall convergence? The Moreau operator was extremely effective at reducing the number of cases required to converge, though this came at a significant penalty in runtime. 1,000 cases with the Moreau operator activated required as much runtime as 60,000 cases without it. Fortunately, the results were much better. With the Moreau operator running on each of 50 population members for 20 generations, the system was able to converge to near zero with the simple constraints activated. Results from this run are presented in Figure 43.

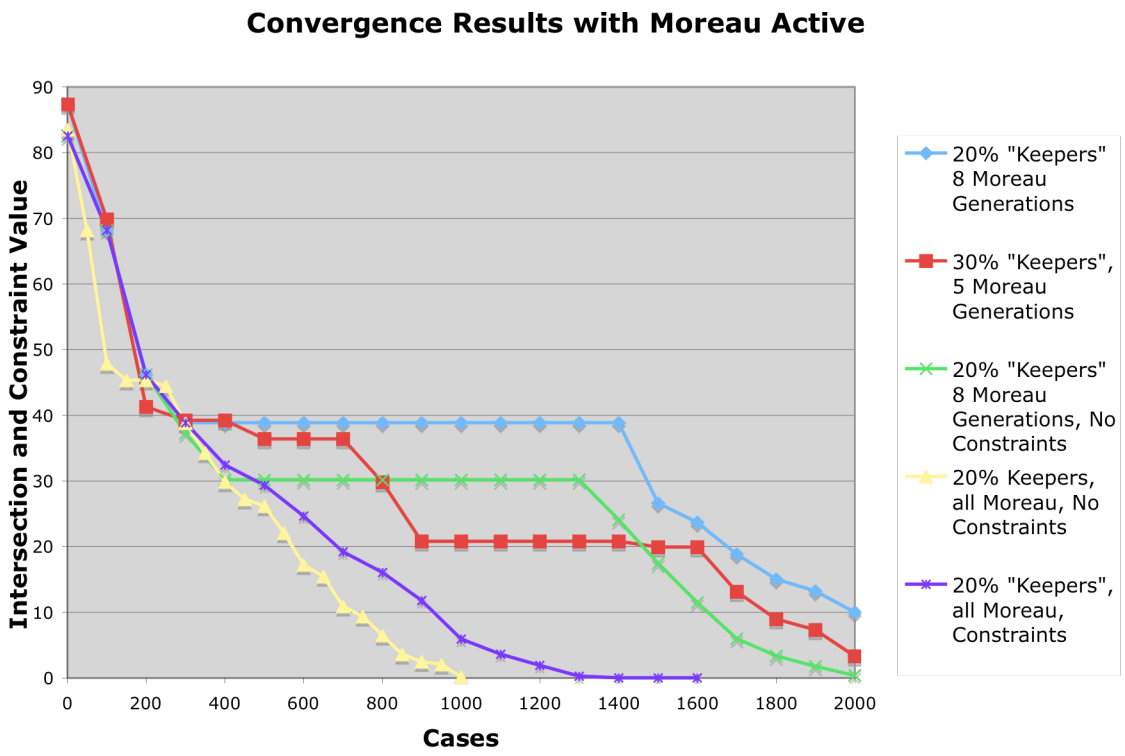


Figure 43: *Convergence results with Moreau options*

With the large amount of time required to run the Moreau cases and the unusual behavior of Moreau-optimized runs presented earlier, there was an opportunity to pursue tuning of the Moreau operator. Cases were run with the Moreau operator active for only part of the optimization. In these runs, the Moreau was active on only five and eight of the twenty generations[xvi]. This allowed a significant time savings, as the eight-generation run required approximately one third the time of the full-Moreau case on the same computer. This was despite the increase in population size from 50 to 100 of each generation

[xvi] Though there were twenty generations in each GA run, only 19 of these were available for the Moreau operator. The first generation of each run has no operations acting on it apart from the act of its initialization.

between the two runs. A reduction in the number of Moreau cases from 49 per generation in the full-Moreau run to 30 per generation in the part-Moreau runs also contributed to the time savings.

The results were not quite as expected. The case with eight Moreau generations actually performed worse than with five generations active. As seen in Figure 43, above, both test runs show significant improvement at points in the optimization when the Moreau operator is active, but the five-Moreau test run shows a dramatic improvement at the eighth generation – a point when the Moreau operator is not active. Investigation of the data shows the improvements in generations eight and nine were due to mutations – simply a case of the GA working as intended.

A final run was made that combined attributes from the full-Moreau run and the reduced-Moreau runs. The larger population size of 100 per generation was used, with the Moreau operator working on 30% of the cases. This arrangement converged to a total value of 0, including both intersections and the simple constraints, within 16 generations. In terms of performance, convergence time was similar to the "full-Moreau" case as a reduction in Moreau operations compensated for the larger population size.

7.3 Constraints Activated

The next step was to create a set of constraints for this arrangement and test them in the context of the existing code base. The initial constraints were by design extremely simple – the point was to demonstrate that constraints could be

integrated into the technique, not to explore a wealth of constraint possibilities and confuse the broader research. The initial attempt involved a desire to bring the center of gravity near the origin. The CG along the x-axis was forced to be at 0, while the CG locations in the y- and z-axis were allowed to vary between +/- 5 units. Intuitively, with a symmetric surface centered on the origin already, this would seem to be simple, as a random distribution of the components would by their nature tend toward that result.

Indeed, the constraints seemed to work well, with few cases falling outside the range allowed. The optimizer also worked well, considering the constraint violations in the same vein as component intersections. In fact, an informal study of convergence with and without constraints initially showed a significantly faster reduction in intersection volume with the constraints activated, though that reduction came at the expense of constraint violation. This, however, was not an effect that continued through the optimization process. The rapid improvement stopped when the value of the constraint violations neared that of the intersection volume, so the effect could have just been an artifact of the additional calculations. Clearly, further study needs to explore the weighting of constraints versus intersection volume, as that will become an important facet of tuning in actual design exercises. At this point in the study, however, the fact that the constraints work is sufficient to continue to the next step.

The constraint activation integrated so well into the overall framework of the optimization scheme that many of the tests were run with the constraints activated when determining overall optimization performance. While this was initially done in error, the small changes in results suggested that results be left as is, since an evaluation of system performance with constraints turned on was

the goal in any event. In Figure 43, for example, the convergence rate for the runs with and without constraints only differ by four generations, or about 25%.

The visual results of activated constraints also demonstrated their effectiveness. While the unconstrained components were placed in chaotic arrangements in Figure 37, components subject to the CG constraints were much better organized. With significantly less space available in the fuselage container than was appropriate to placement with the constraints activated, the optimizer was forced to choose more organized arrangements. This can be seen in Figure 44. Much of the container has been left empty as the components were placed against one wall in order to maintain the CG limits set within the optimizer. Further, unlike the components in Figure 36, these components were placed without limits on rotation; the optimizer chose the relatively square alignment.

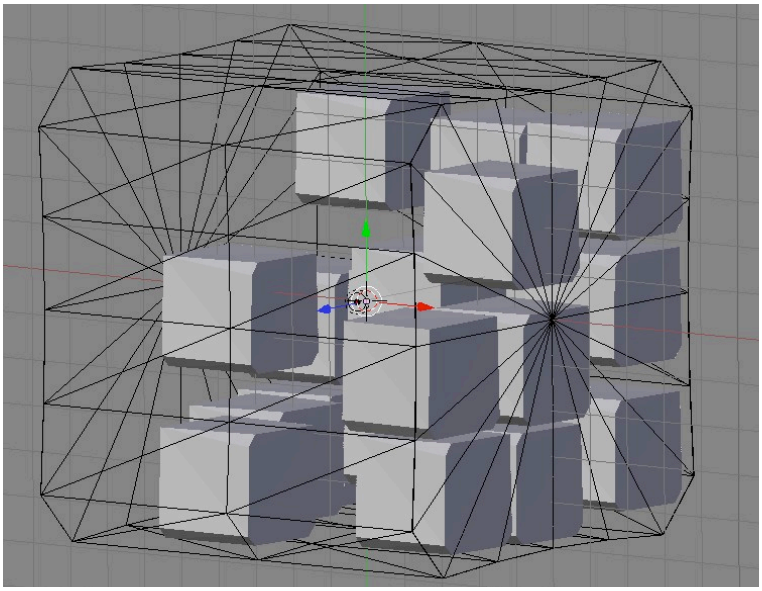


Figure 44: *Components placed with constraints*

7.4 Robustness and a Cost Model

The idea of a probabilistic analysis of volume comes from an intent to quell the risks associated with uncertainty and produce a robust design that will still work well in the face of a wide range of changes in component size. In any design, failure to understand uncertainty can lead to missed performance targets, cost overruns, or a failure to meet other desired goals. In this case, uncertainty about the actual sizes of individual internal components can generate in a poor fit in analytic models, which can then result in an aircraft that cannot perform its missions.

To evaluate the effects of uncertainty in volumetric calculations, a small set of experiments has been devised. Here, no aircraft are used in the test; integrating these ideas into an aircraft design will be addressed in Chapter 8.

A simple container is designed to hold the target components. Not quite a rectangular box, it features small angled cutouts and curved detents in order to provide the optimizers unusual spaces for placing objects. The components to be placed are designed to not quite fit in their standard configurations. They too are mostly rectangular boxes with additional cutouts to simulate truly unusual shapes. Four boxes nearly fitting in such a container as optimized by the code developed for this research are visible in Figure 45.

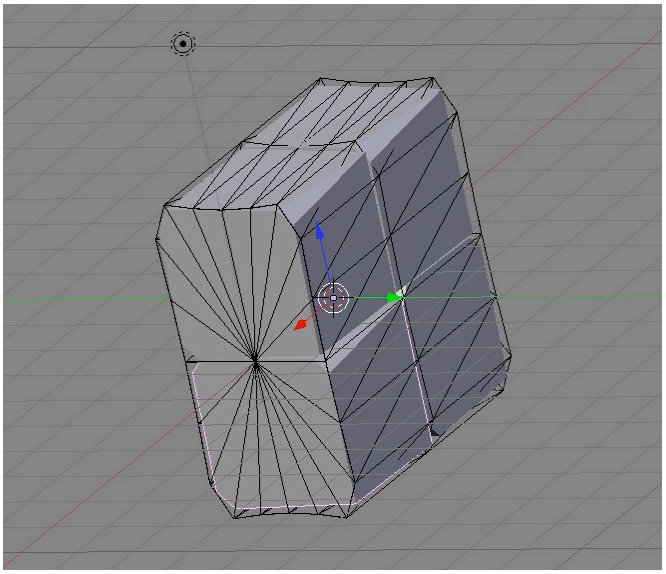


Figure 45: *Overly packed container as optimized by the GTS/C code*

Each component has associated with it a range of possible sizes. At this point, there are several ways to compute the effects of component sizing on the overall design. The first is to simply assign a distribution to the component sizing and allow it to be controlled as a variable. This would, for example, produce nearly a zero percent chance of the components being built at their smallest volumes and a 100% chance of being built at the maximum volume. This represents a uniform distribution of uncertainty about the size of the components.

7.4.1 Cost and Viability

There can be a cost associated with counteracting the uncertainty in this model. If, for example, the components are used at their default sizes, there is no additional cost associated with designing the components. If the components need to be used at smaller sizes, the cost of acquiring the components increases.

This cost may represent the true cost of sourcing a smaller component with equivalent functionality, or it may represent the cost of redesigning the component to meet the specifications. The uncertainty is then handled as an exploration of the costs themselves. At what point is it smarter to redesign the container compared to redesigning or replacing a component?

At the other side of the equation lies the performance of the system. In an aircraft system, if the components cannot fit, the aircraft will need to grow in size to accommodate them. For an aircraft, there is also a performance cost associated with the modifications to the airframe. In this case, where the system is much too simple to represent an aircraft, there is a cost associated in redesign (though it should be small since the system would still be in the conceptual design phase). The performance cost of changes might be represented through the need to redesign accessory items. In the next chapter, the performance costs of such a modification will be explored in detail with an aircraft example, but here, a basic manufacturing analogue was used on this simple system.

7.4.2 Exploration of a Simple Problem

If the container used in this sample problem represents an existing system, and the components put into it represent some notional objects that are required to fit into it, there will be a cost associated with modifying either of them. In the case of the components, this cost, as described above, comes from outsourced purchasing or design of a new component. In the case of the container itself, the cost comes from construction of new tooling and the manufacturing expertise required to produce the container or to modify it.

The model involved placing four components into the container. The components and the container had a default cost to manufacture at their "normal" sizes. If the components all fit, there would be no additional cost associated with the manufacture of the unit, simply the default production costs. If the components did not fit, there must be a compromise between the cost involved in remanufacturing the container with that associated to redesign of the components themselves.

If the components needed to be redesigned, they would have a cost associated with that based on the reduction in size. As a small offset, smaller components had a corresponding cost reduction for manufacture to reflect a reduction in materials used. This functions in the same manner as an airframer who redesigns a part to reduce the amount of aluminum required to make it. If, on the other hand, the container is redesigned, there is also a change in cost associated with the redesign. This cost has no advantageous manufacturing offset, as the container is always enlarged to fit the components.

Cost factors for all of these variables were controllable within a simple *Excel* environment designed for this study, with places for the quantity of containers made as well as design and manufacturing coefficients for each part. A sample view of the *Excel* environment is shown in Figure 46.

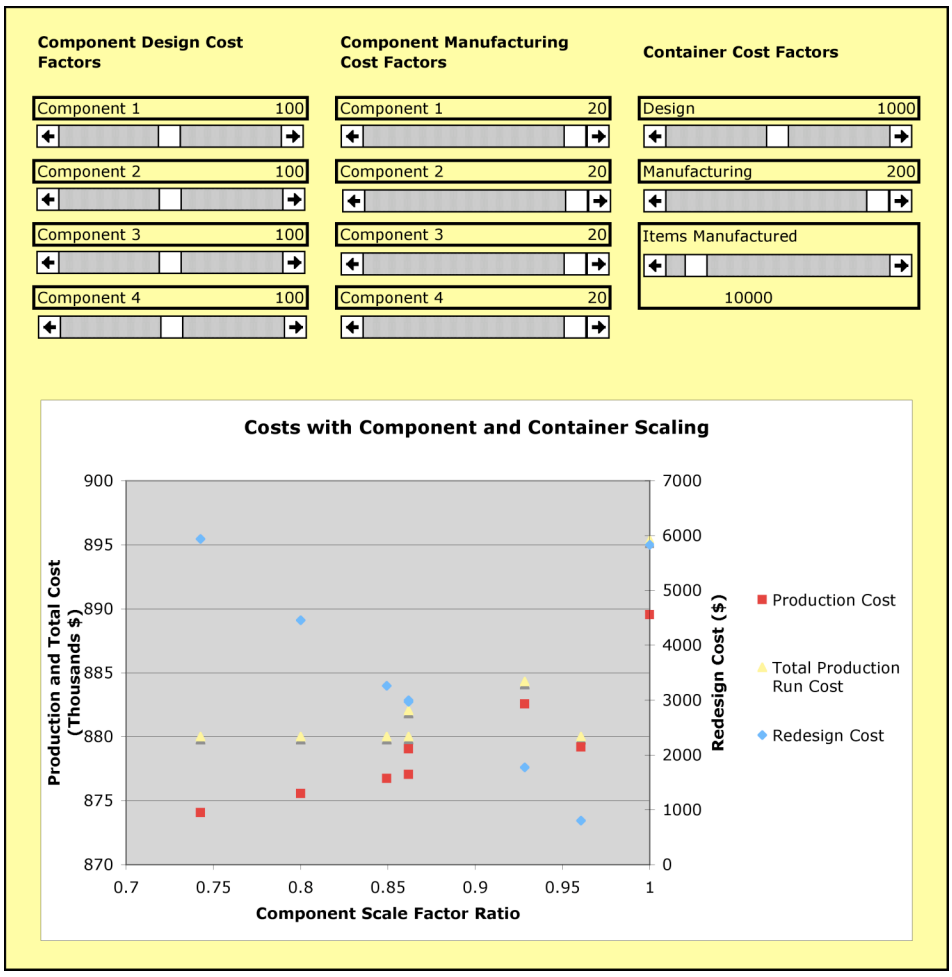


Figure 46: *Simple Excel cost environment*

The *Excel* environment made use of a simple cost structure similar to that used by NASA's *Aircraft Life Cycle Cost Analysis Tool* (*ALCCA*)[149]. Cost for this simplified model was initially broken into design and production costs. Design costs only applied to designs that were changed from the default size. When they were applied, design costs were based on the amount of change from the default size, so a reduction in component size would result in a design cost, while an increase in container size would also result in a design cost. Production costs were assessed whether a component or the container was changed or not.

The first step in the process was the evaluation of the options in the design space. The container space provided in this case is too small, as seen above, producing overlap with any arrangement of components. This forced at least some redesign of either the components or the container that holds them. Within the *Excel* environment, the design and production costs of the components and container could be varied easily; these values were combined with the volume of intersection for each possible configuration of redesigned components. The cost values were plotted in a chart attached to the tool, with points indicating design, production, and total cost for a run of products. The results were plotted with cost (in dollars) on the y-axis; the scale factor ratio, showing how much the individual components have been shrunk in order to fit properly in the container.

Results here represent a test problem designed solely to test the basic methodology; no numerical results are presented here. Results for an actual product design would differ based on the cost structure of the individual components. A full exploration of the economic modeling possibilities within volumetric sizing is left for future work, though a simple cost model is presented in the final demonstration problem. While this problem was designed to be only exploratory in nature, it does demonstrate that volumetric awareness can be incorporated in decision-making early in the design process.

8 Final Demonstration Problem and Results

The final measure of this work's success is the application of the methodology to an actual aircraft design. To be successful, the proposed intelligent, robust method should be able to reduce design uncertainty and improve the quality of the final design by bringing internal layout forward in the design process.

As mentioned earlier, the best application for volumetric sizing is in aircraft that are tightly packed, tightly constrained, or both. This argument leads almost immediately to supersonic aircraft, particularly fighters, and to uninhabited aerial vehicles (UAVs). Most fighter information is either classified, of little practical value in the context of this work, or out of date. This makes data for comparison nearly impossible to find for these aircraft. While internal layout information is available for the early model F16, for example, it is an aircraft that is now thirty years old[150]. The most recent cutaways presented in Chapter 1 are now nearly ten years old. Similarly, no supersonic transport aircraft have been manufactured since the Concorde, so that information is also out of date, and new designs for aircraft such as the High Speed Civil Transport (HSCT) and supersonic business jet (SSBJ) have not progressed beyond the preliminary design phase.

The nature of UAVs means that publicly available information is difficult to obtain, and available information is also often proprietary. UAVs are primarily used by military and government entities for surveillance and remote attack, or for other surveillance-related activities such as weather monitoring, so the

closely-held nature of most detailed UAV data is unsurprising. Fortunately, the UAV market is far more robust than the market for supersonic aircraft. Indeed, in 2005, the American Institute of Aeronautics and Astronautics listed more than 400 UAVs either in service or in final development[151].

One unusual aspect of UAV operation is the often-unpredictable nature of their use in the field. While a vehicle may be designed for a particular payload package, this payload may be superseded almost immediately by something larger, heavier, or both[152]. This forces UAV designers to reconcile a relatively wide range of payloads and payload locations – a quality well-suited to this research. This also provides a rich area for the exploration of a robust design sufficiently flexible to consider a range of payload possibilities.

To begin the test problem, a notional UAV was designed as a test platform. This aircraft was initially created in *X-Plane*, with structural weights estimated from an existing airframe of similar size and configuration. *X-Plane* telemetry was then used to create a set of drag polars and an engine deck for use in *FLOPS*[153]. With a *FLOPS* file created, rapid analysis of a host of mission parameters was possible with the notional aircraft. A simple cost regression added depth to the analysis possible with this volumetrically-driven model.

A fuselage container space was generated to approximate the space available for fuel and payload in the aircraft. A range of volumetric studies was performed on the aircraft to determine which arrangements were possible within the constraints provided for CG and placement. These studies included explorations of changes in the customer payload size and weight as well as changes in the other internal components. The consequences of poor fit were then explored through manipulation of the *FLOPS* model based on the results of

the fit checks. Aircraft performance with a wider versus a longer fuselage was compared to determine the results of changing the consequences of a poor fit. A simple cost model was also implemented to simulate the business decisions that can be made in the design process when volumetric sizing is incorporated in the early phases of design.

These features combine to produce a design that changes based on customer needs. Some customer needs would be difficult to accommodate without the use of volumetric sizing. In this case, most of the difference comes in the form of changes to the customer cargo. While a UAV may be designed for a particular payload, what happens if the customer decides to double its size? What about an increase in payload volume of three or four times? What will that do to the design? Further, what if the fuselage is well within its limits for the cargo volume but other internal components are larger than originally predicted. What do these changes mean to the final design?

8.1 Baseline Vehicle

The baseline vehicle for this demonstration is a notional aircraft. Though inspired by existing UAVs, no UAV with these characteristics exists as far as the author is aware. The initial model began with an outer skin shape, physical dimensions, and propulsion system all defined in *X-Plane*. Next, a mission analysis was performed in *FLOPS* using a simple, endurance-based flight profile to evaluate the aircraft's capabilities. The initial model and mission performance

provided the traditional design parameters necessary to size the vehicle and became the baseline for further comparisons. Since this vehicle served only as a demonstration design, the mission could be sized to the vehicle rather than the traditional sizing of the vehicle to the mission. An internal layout analysis completed the baseline aircraft.

Many UAVs in current production or development share a common configuration layout: a relatively short fuselage with a pusher propeller, flanked by twin booms that support an inverted "V" tail configuration. This configuration has the advantage of providing room at the front of the fuselage for cameras used for piloting and observation. The inverted "V" tail configuration also keeps the tail surfaces out of the propeller slipstream, making the aircraft's handling qualities less likely to change drastically with changes in engine output. The configuration is also generally simple to assemble, as body components are not usually complex and can often be manufactured separately and then assembled late in the construction process. Assembling the aircraft from prefabricated components also reduces the need for large autoclaves for composite curing and encourages the use of outside manufacturing sources, allowing the UAV manufacturer to concentrate on their areas of expertise.

A model inspired by several real aircraft has been chosen as the sample problem for this study. Using the generic UAV layout described above, the sample aircraft was designed to carry a 10lb payload of sensor electronics on missions lasting more than 24 hours. Cruise speed was approximately 55 knots, with a low speed loiter just below 50 knots and stall at less than 40 knots according to performance tests within *X-Plane*. The aircraft was intended to be operated semi-autonomously, with ground-based operators giving the aircraft

guidance for mission destinations and interceding when necessary for a particular mission requirement. Additional common UAV features such as a parachute recovery system were not used in this study.

The aircraft external design was generated in *X-Plane*. Weight fractions were projected from values collected from a flying fuel cell UAV prototype aircraft built by a team from Georgia Tech's Aerospace System Design Lab (ASDL) and the Georgia Tech Research Institute (GTRI)[154]. Significant specifications for the two aircraft are shown in Table 7. While the UAV designed for this study was significantly heavier than the fuel cell demonstrator aircraft, its different powerplant characteristics and smaller wings resulted in an empty weight similar to that of the fuel cell demonstrator.

Flight performance was also analyzed within *X-Plane*, with the simulator utilized for primary point performance characteristics and for an estimate of handling qualities.

Table 7: *Fuel cell prototype and notional UAV specifications*[155]

	FC Demonstrator	Notional UAV
Wing Area (ft^2)	20.2	13.4
Aspect Ratio	23.0	12.2
Span (ft)	21.6	14.2
Taper Ratio	0.7	0.3
Tail Area (ft^2)	4.9	2.4
Length (ft)	7.8	8.5
TOGW(lb)	36.2	65.0
Wing Weight (lb)	6.3	5.0
Tail Weight (lb, incl. booms)	2.2	2.1
Cruise Speed (kts)	28.2	56.0

The mission performance analysis was conducted within *FLOPS*. Aerodynamic drag polars and engine performance decks were exported from *X-Plane* through *Excel* spreadsheets and formatted for inclusion in the *FLOPS* input file. A baseline mission with the *FLOPS* model was then flown to evaluate the notional aircraft's performance. The baseline mission was relatively simple: takeoff, climb to a cruise altitude, cruise a set distance at the velocity for best range, loiter on target at the speed for best endurance as long as possible, then

return to the launch point and land. A graphical representation of the mission profile is shown in Figure 47. This model produced a starting point for analyzing consequences of any future volumetric choices.

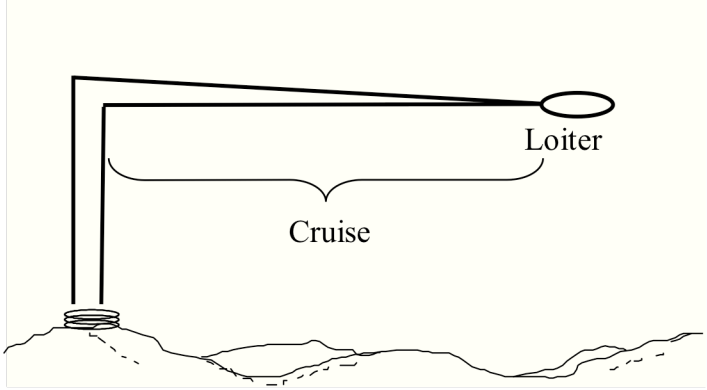


Figure 47: *Test UAV mission profile*[156]

The volumetric concerns were now considered through the creation of a fuselage compartment to hold the sensor equipment, electronic controls, and other items. Components were also generated at this time, along with ranges of uncertainty for their sizing. Having a range of uncertainty for components within the aircraft allows analysis of their fit over a range of sizes – predicting changes in customer needs later in the design process. The fuselage container and all included components were created using the *Blender* visualization software[157]. Component constraints for center of gravity, local placement/orientation, and distance from other objects were also provided at this point. Restrictions in component placement and overall aircraft CG range, for example, will help guide the results.

8.1.1 X-Plane Model

The model created in *X-Plane* is of a notional aircraft, but the consistency of flight characteristics modeled in *X-Plane* should provide a sufficient basis for aerodynamic performance of an aircraft of this size and weight[158]. *X-Plane* has been successfully used by the author on a number of design projects. In a study of potential retrofitting of a Cessna 172 with a fuel cell-based propulsion system, *X-Plane* was able to generate performance figures within 5% of the published values in every tracked measure[159]. In a study of a small UAV in cooperation with the Georgia Tech Design-Build-Fly team in 2002, the *X-Plane* model successfully predicted both mission performance and handling characteristics. A picture of the notional aircraft is provided in Figure 48. Aircraft external geometry was simplified in order to isolate effects of the internal layout.

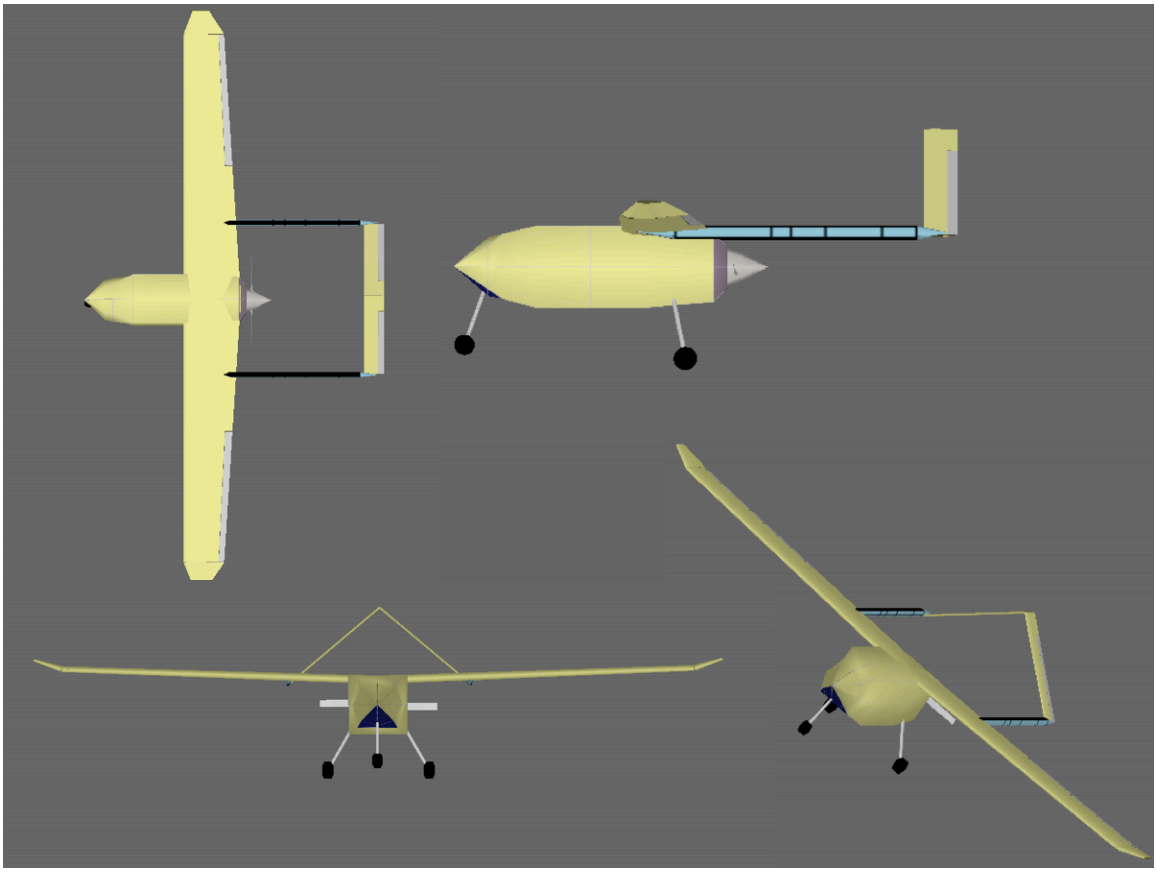


Figure 48: *Views of the test UAV from* X-Plane

As mentioned above, aircraft weights were estimated from a comparison with the Fuel Cell Demonstration Aircraft. This also produced the implicit structural relations needed by X-Plane to model an accurate aircraft. As the aircraft was only modeled for demonstration purposes, no further structural analyses were performed.

X-Plane differs from most commercially available flight simulators in its fundamental modeling scheme. Rather than modeling an aircraft as a point mass with an associated set of performance derivatives, *X-Plane* takes a more physical

approach to flight. Flight in *X-Plane* is modeled as a force balance among all of the primary components of the aircraft. Lifting surfaces contribute lift, drag, and moments, while "body" components primarily contribute drag (though small amounts of lift are possible for some shapes). The total forces generated by each component at a particular flight condition are combined into a sum that may result in anything from straight and level flight to a catastrophic change in flightpath[160].

The calculations used by the application are at a similar level to those taught in undergraduate performance classes. Empirical relations are used to calculate airfoil efficiency and airframe drag, for example. Two-dimensional (2D) airfoil data is recorded in a format similar to that used by Abbott and Von Doenhoff in their *Theory of Wing Sections*[xvii,161]. This 2D data is then modified based on the aircraft's wing aspect ratio, twist, etc. to generate a series of lift, drag, and moment coefficients for the entire wing.

The basic test vehicle used in this study makes use of a NACA 2412 airfoil. The 2412 is a popular member of the NACA Four Digit Wing Sections[162]. Two separate 2D curves were used to model performance at a wide range of Reynolds numbers. The wide-span ailerons visible in Figure 48 can also droop slightly as flaps to increase lift and drag at takeoff and landing. The tail is based on a NACA 0009 airfoil, and its control surfaces are mixed in such a way that they can control pitch and yaw. The engine is modeled as a 5hp reciprocating engine with a fixed-pitch two-bladed propeller using a Clark Y[163] airfoil section.

[xvii] The author has used airfoil data from *Theory of Wing Sections* in *X-Plane* to good effect on several prior projects.

Several test vehicles were modeled in *X-Plane* in anticipation of alterations to the final aircraft in concert with changes in the internal layout. Models with extended fuselage lengths and larger diameters provided drag polars for a variety of *FLOPS* studies. An explanation of the performance analysis in *X-Plane* is detailed below. Since the engine was unchanged throughout the study, the engine performance values used to generate an engine deck in the initial tests were used throughout the remainder of the *FLOPS* runs, described in the next section.

8.1.2 FLOPS Model

The drag polars produced by *X-Plane* were then input directly into a text-based *FLOPS* input file, bypassing the program's dedicated aerodynamics module. Unlike *X-Plane*, *FLOPS* does model an aircraft as a point mass with associated aerodynamic characteristics[164]. Though *FLOPS* does possess a set of aerodynamic regressions, previous work by the author has found them to be ill suited for aircraft in this size and velocity range. *FLOPS* is also capable of calculating component weights using regressions, but for the purposes of this study, a fixed set of weights was entered. Weights for most components remained constant, with the only changes made being those suggested by scaling of the internal components and fuel burn. The engine data from *X-Plane* was converted into a *FLOPS*-readable engine deck. Having a standalone engine deck allowed any number of test aircraft to be modeled using the same propulsion system.

The mission developed for the test aircraft was simple: takeoff at sea level under standard atmospheric conditions (and no wind), climb to 2,500ft, fly out

80nm at the velocity for best range, loiter on target for as long as fuel allowed at the velocity for best endurance, return at the velocity for best range, and land at the original point of departure. *FLOPS* generates a variety of output, but for the purposes of this study, maximizing the time on station during the loiter phase of the mission was the greatest measure of success for the aircraft. With a single value, time on station represents a rapid way to evaluate an aircraft's aerodynamic and propulsion system effectiveness in the context of the simple mission described above. This single metric was not viewed completely in isolation, however. Other concerns, such as whether the aircraft could attain the proper cruise altitude during all phases of the flight, were also checked to ensure consistent results across design changes.

8.1.3 Internal Layout and Component Fit

Neither *FLOPS* nor *X-Plane* has any capability of dealing with internal layout. *FLOPS*' only explicit concession to volumetric concerns is a simple check of fuel weight capacity, which it links to fuel volume capacity. Similarly, *X-Plane* is only concerned with the outside of the aircraft. Lacking any internal considerations, *X-Plane* only places items that might contribute directly to the aerodynamic performance of the aircraft. *X-Plane* models created by the naïve or by users intentionally generating unrealistic aircraft can violate many physical limits with respect to aircraft structure. For this reason, the weights used in the fuel cell demonstration aircraft acted as a guide for the models used here.

Code developed for this research was then integrated with data from *X-Plane* and *FLOPS*. The basic *X-Plane* outer fuselage lines provided a limit for internal space. Wings and tail booms were not considered for this study in order

to better focus the research. Aircraft in this size range often have few components placed in the wing and booms, so this is a realistic approach. A fuselage internal cavity for the aircraft's internal components was created using GTS and *Blender*. Since the engine placement was trivial due to its well-defined placement in *X-Plane*, the fuselage container only concerned space forward of the engine firewall. The engine area was considered to also house the start battery and power generator to power the sensor load.

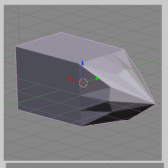
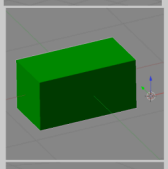
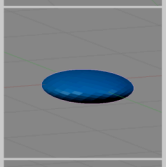
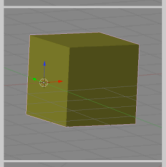
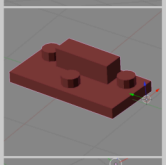
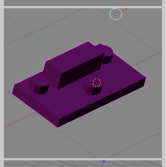
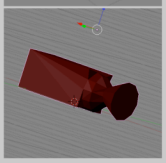
The components themselves were now developed. While a UAV may carry any sensor load a customer may request, a relatively simple load was developed for this aircraft. As an autonomous aircraft, the UAV required a forward-facing camera. An additional forward-facing camera was also installed to view in the infrared. These cameras were both required to be placed in the nose of the aircraft to be able to see properly, and these absolute position constraints operated as a test for the code's optimizers. The aircraft also required a computer box for the autopilot system. As a large source of heat, the computer box could not be placed near the infrared camera. This provided a relative position constraint for the optimizer. A pair of radio transceivers was also required for navigation and communication with the ground station. Two GPS antennae were also required, and these were placed at the top of the aircraft to maximize exposure to satellite signals. A pair of "mystery boxes" representing specialized customer-required cargo were also placed in the aircraft. Finally, a non-integral fuel bottle with relatively simple shape was included. This fuel bottle, in addition to replicating the type of fuel system used in the fuselage of several UAVs now flying, allowed the CG of the aircraft to be tailored to each

mission with only a slight shift in longitudinal position of the fuel tank within the fuselage.

Most of the components in the aircraft were considered to be of fixed shape. This represents the desire to use commercial, off the shelf components for cost reasons. Each of the customer cargo components was assigned a size range that represented the certainty of obtaining a component of that particular specification. This size distribution was presented as a simple photographic scaling of the entire component and involved increasing the cargo volume in steps up to four times its original value.

Individual components were now checked within the GTS-based environment described above. This helped determine the probability of fit without violating any constraints at a range of several different component sizes. Checking for fit at different component sizes accounted for the uncertainty inherent in their description. The results were then fed back into the *FLOPS* models, where modified versions of the aircraft can be evaluated for potential performance improvement/degradation. A list of the constraints used in the internal layout analysis is provided in Table 8.

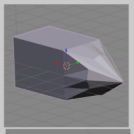
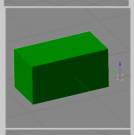
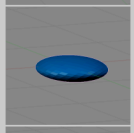
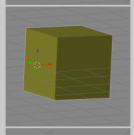
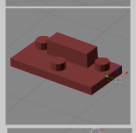
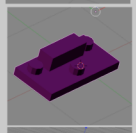
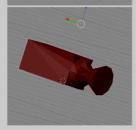
Table 8: *Shape constraints for final test problem*

	Shape	Number	Constraints
	Fuselage Container	1	Must Maintain overall CG within acceptable range
	Customer Cargo Box	2	None. Will vary in size/weight with customer demands
	GPS Antenna	2	Must be at the top of the fuselage
	Fuel Tank	1	Can only move along the longitudinal axis of the aircraft
	Transceiver	2	None
	Autopilot	1	Generates heat – must be kept away from infrared camera
	Camera	2	One camera is infrared sensitive (see above). Both must be in the front of the aircraft at the bottom

8.2 Initial Fit Trials

The GTS-based environment now needed a set of components to test in the fuselage container. Initially only the simplest shapes were demonstrated in the environment. The fuel tank and cargo boxes were both modeled as rectangular boxes, so they posed no difficulties. The GPS antennae, while made up of a large number of points, and therefore computationally large, were still relatively simple shapes and similarly posed no problem for the fit environment. The radio transceivers were built from several simple shapes combined in Blender and exported into GTS files. The transceivers, cameras, and autopilot initially had problems integrating with the environment as mentioned below, but the concerns were resolved by slightly modifying the shapes. These modifications had no negative effect on later analyses. With the conversions operating properly, the transceivers were integrated with the remainder of the shapes. The cameras involved significantly more manipulation before they could be used. Eventually all the required components were integrated into the fit environment. A list of the surfaces used in the final test problem and their specifications is provided in Table 9.

Table 9: *Surfaces used in test problem*

	Shape	Number	Length	Width	Height
	Fuselage Container	1	33	12	12
	Customer Cargo Box	2	9.7-12.3	5.0-6.25	5.0-6.25
	GPS Antenna	2	5	5	1.6
	Fuel Tank	1	11.4	11.6	11.7
	Transceiver	2	8.9	4.4	2.3
	Autopilot	1	8.4	4.2	2.2
	Camera	2	7.9	2.8	2.8

All measurements in inches

8.2.1 Tests of early shapes

Initial tests were performed primarily to ensure that the shapes themselves would have no incompatibilities with either the code environment or each other. During this phase of testing, incompatibilities were discovered between several of the shapes and the GTS environment. One cause of the problem was found to be the conversion between *Blender* and GTS file formats

and the specific requirements of the GTS fit environment. This is described in detail below. With the shape incompatibilities eliminated, placement continued as planned. A few unusual problems were discovered that were related to the significant difference in size between the fuel tank and several of the other components. Component intersections between small objects and the fuel tank were not being calculated when the components were completely immersed in the tank. In this problem, none of the components were allowed in the tank, but in earlier work the components had been so similar in size that a complete containment was not possible, so an intersection value related to these inclusions had not been developed. These problems were also reconciled through the addition of an intersection model for the tank inclusions, and the remainder of the tests concentrated on fully developing the fit and constraints in preparation for the final cases. A final, more serious set of problems centered on issues within GTS that were resolved by slightly modifying the component files. These problems are discussed briefly in Section 6.2 as well as in detail in Appendix B.

Exercises oriented simply around fit demonstrated that the collection of internal components would fit in the fuselage container without the constraints activated. Further exploration with a simplified set of constraints demonstrated that the constraint calculations themselves worked, but the initial set of tests did not evaluate component fit with all of the constraints set to their final values. Later tests included all final components as well as the final constraints.

8.2.2 Elimination of shape incompatibilities

The cameras and autopilot used in the internal layout model were created from a set of simpler shapes that were joined in *Blender* in much the same

manner as the other complex shapes. Unfortunately, the interface between the individual shapes that made up these components was not as simple as the interface in the transceivers and other components. The final composite shape produced by the file conversions was not usable.

The difficulty originated in the basic manner in which GTS represents surfaces. Composed of sets of vertices, edges and faces, GTS surfaces can be as arbitrarily complex as desired, but they have no fundamental requirement to be physically possible. This characteristic speeds the modeling of components in applications such as video games where only the appearance of physical possibility is important. In order to work effectively with any code involving physical shapes, such as that designed for this research, however, every GTS shape should have a straightforward description of what is "inside" and what is "outside". In most cases, such as the triangles used to construct a simple sphere, this is not an issue, as the direction of "inside" is simple to calculate from the surrounding surfaces.

While this seems intuitive, the conversion between file formats with complex shapes can result in individual triangles being placed, quite literally, with the "in-side out". Unfortunately, in the case of the modeled cameras and autopilot, some of the faces making up the surfaces had become reversed in the conversion process. This resulted in shapes that considered the same direction to be both the inside and outside of the surface. A physical impossibility, this contradictory shaping had to be eliminated before the tests could continue. Eliminating surplus vertices, edges and faces from the shapes removed the inconsistencies and allowed use of the cameras and autopilot in later fit checks.

8.2.3 Final fit test environment

The final fit test environment included all ten components and the fuselage container within the GTS/C code developed for this work. The first step involved determining the quality of the baseline fit: would the existing components fit in the fuselage container without violating any of the constraints? In this case the answer was yes, though the optimizer nearly ran into its run limit before converging. While the components would fit easily on their own with no limits, the constraints added enough restriction to their placement that the optimizer was challenged to discover a set of coordinates that would not show intersections, constraint violations, or both. Future growth was also limited, resulting in a design that would be sensitive to potential changes in payload size.

The long and wide fuselage containers were both easily large enough to contain all of the components without violation of constraints. The surplus space resulted in aircraft that were robust to changes in the size of the payload and equipment carried. But were they optimal? The answer here was no. While sufficient space was available in each of the altered fuselages for all of the components, they possessed a surplus of space that in a conventional design would contribute nothing but drag to the final analysis. In this case, the ability existed to explore other alternatives. First, however, the flight and mission performance of these variations needed to be analyzed, and the consequences of changes to the fuselage explored.

8.2.4 Fit examples

The component fits progressed quite predictably in the GTS/C environment. Initial guesses by the optimizer were not very good, and inspection

of the intermediate results showed that some of the arrangements selected in the earliest generations contained a large amount of overlap and constraint violation. Later tests with large cargo and component sizes also showed components on the outside of the fuselage initially as the intelligent placement algorithms were sometimes shut off due to difficult placement. An example of a poor early fit is seen in Figure 49, where the fuel tank is located outside of the fuselage – hardly the desired configuration.

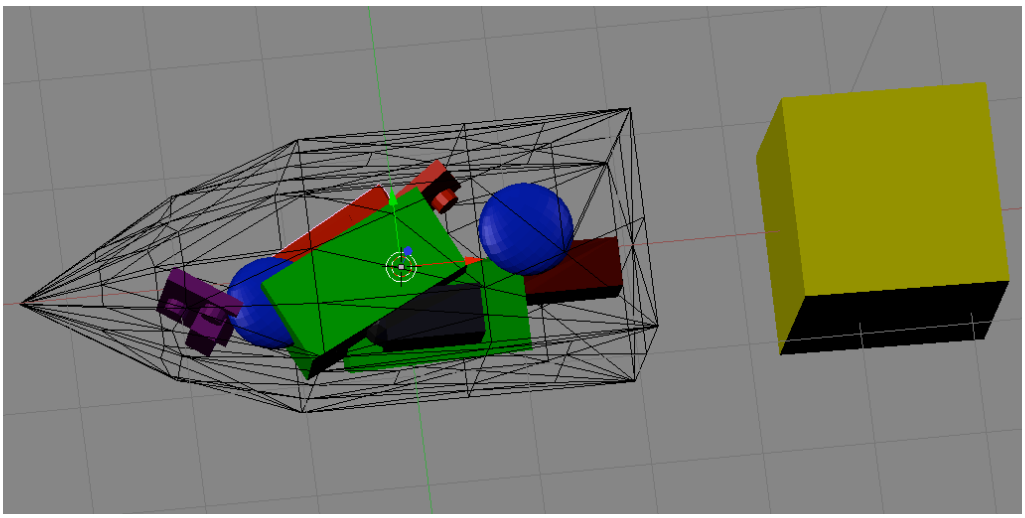


Figure 49: *Component fit after one generation*

As the optimization progressed, the fit became better in most cases. Some cases still violated constraints and produced poor results, but these cases were not allowed to remain in the population base. The most successful cases involved repeated applications of the Moreau operator, though closer investigation of the data showed a number of successful cases being formed through mutation of an already good case or through crossover between two good cases. As the

optimization progressed, cases that sometimes appeared good actually had problems, and the generation size was enlarged for cases that were "tighter", and had less surplus space. The sample in Figure 50 has a fairly significant intersection between two components (the red transceiver is piercing one of the green customer cargo boxes), yet the remainder of the layout is good, so this arrangement was the best at the end of 2,000 cases tested. Doubling the population size of the genetic algorithm and expanding the number of generations by 25% eliminated cases like this.

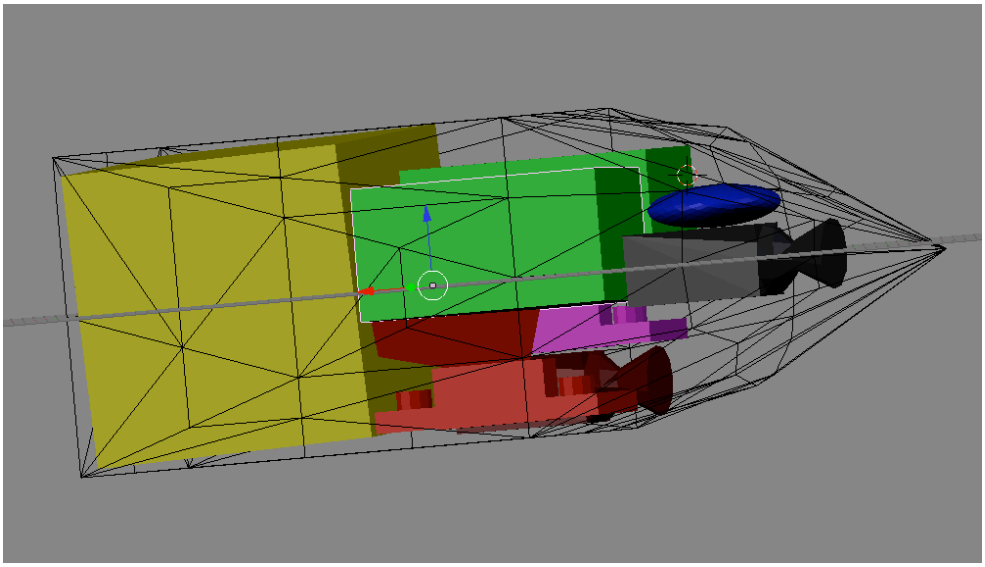


Figure 50: *Improved layout with intersections at 2,000 cases*

Once the default shapes could be fit properly, the study turned to investigating the largest size customer cargo that could be fit in the three fuselage shapes. The baseline fuselage was investigated. The default cargo boxes fit, as did the double-sized cargo, though convergence was slowed significantly. The triple cargo fit the long and wide fuselage but was too large for the baseline.

The wide fuselage was the only configuration capable of fitting the quadruple-sized cargo. Table 10 shows the fits.

Table 10: *Fuselage cargo fit options*

		Fusleage Choice		
Relative Scale	**Volume (in^3)**	**Baseline**	**Long**	**Wide**
1	241.9	fits	fits	fits
2	483.8	fits	fits	fits
3	725.8	**no**	fits	fits
4	967.7	**no**	**no**	fits

8.2.5 Aircraft point performance and aerodynamic results

The first step in the determination of aircraft flight performance was to fully explore the *X-Plane* model. Once created, the baseline aircraft was flown at a range of flight conditions to simulate a real mission. Takeoffs, landings, and standard maneuvers contributed to an informal assessment of handling qualities. More structured maneuvers were then used to generate the drag polars and engine deck information.

The engine deck data was relatively simple to gather. Data recording began on the ground, where a variety of power settings were tested to determine sea level static thrust at a range of throttle settings. Atmospheric conditions were set to those of a standard day with no wind using *X-Plane's* weather capabilities.

The aircraft was then flown at several select altitudes to generate an idea of engine/propeller performance at a range of altitudes and Mach numbers. Much of the aircraft's flight envelope was explored, with altitudes to 16,000ft and velocities up to Mach 0.12[xviii]. *X-Plane's* data output function was used to record the resulting telemetry in a large, delimited text file that was then processed through *Excel*.

Aerodynamic data was recorded in much the same way. With the aircraft in straight and level flight, the autopilot was used to maintain heading and altitude while the engine power was changed over the full throttle range. This forced the aircraft to adjust angle of attack in order to maintain level flight, generating a set of curves that then produced a drag polar to compare lift and drag in their relationship with one another. The telemetry data was again exported into *Excel*, where it could be manipulated easily.

The *Excel* manipulations themselves reduced the data significantly by retaining only flight data that fit several criteria. A small rate of climb was required, as basic performance calculations become much more difficult when the aircraft is not flying straight and level. A nominal lift value was also important, as lift values either significantly higher or lower than the weight of the aircraft indicated undesirable maneuvers or portions of the flight when the aircraft was actually on the ground. Finally, slip was reduced as significant yaw disturbed the aerodynamic performance of the aircraft. The three limits are shown below in Table 11.

[xviii] Engine deck data did not extend as high as the maximum Mach number obtained in the simulated flight tests, as the transient nature of flight conditions at those velocities cast doubt on the usability of the data.

Table 11: *X-Plane flight test performance constraints*

Flight Constraint	Lower Limit	Upper Limit	Units
Vertical Speed	-200	200	ft/min
Total Lift	55	75	lbs
Slip Angle	-1.2	0.5	degrees

Flight data was recorded for the baseline aircraft as well as for a "wide" aircraft, with the fuselage width and height each increased by 3.2 inches and a "long" aircraft with the fuselage stretched by a foot. This produced an increase in the fuselage container volume of 60% and 36% over the baseline aircraft, respectively. Engine data was only recorded for the baseline vehicle since the engine did not change between aircraft and additional interference effects caused by the fuselage changes are outside the abilities of *X-Plane* to calculate.

The flight results were similar to what was expected. The lift curve slopes for the aircraft were nearly identical, as shown in Figure 51. The wings were not changing among the three aircraft, so this served to help confirm the consistency of the *X-Plane* environment. A good deal of noise in the data exists at smaller angles of attack. This effect may be due to numerical noise in *X-Plane* or transient effects from maneuvers at the relatively high speeds normally associated with flight at such small angles. Noise also exists at high angles of attack, but this is primarily due to the difficulty in flying straight and level in post-stall conditions. The long aircraft, interestingly, was most difficult to control at high angles of attack, reducing the quantity of data produced at those areas of the flight

envelope. As none of the mission was flown in this area of the curve, this did not adversely affect the results.

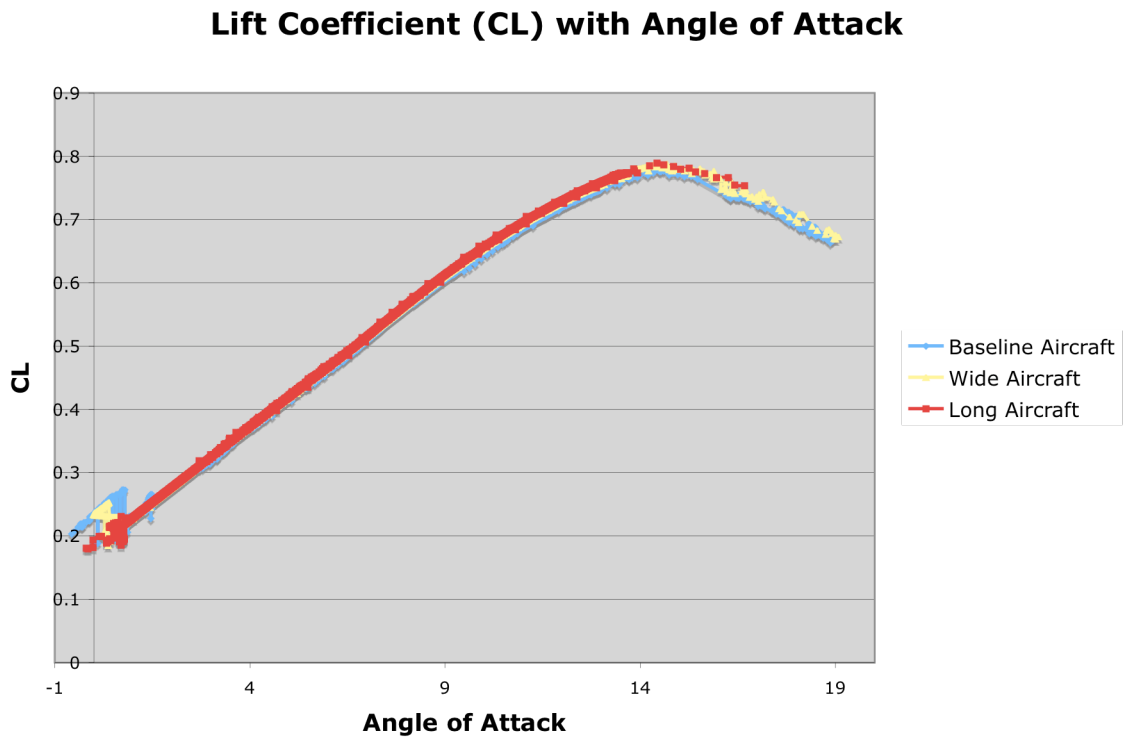


Figure 51: *Lift curves for three test aircraft configurations*

Similar results were created when exploring the drag polars themselves as shown in Figure 52. In this case the relationship between lift and drag for the baseline and long configuration were quite similar, though the baseline configuration did result in a maximum lift/drag slightly higher than that of the long configuration. The wide configuration here shows the effects of increased surface area. Drag is noticeably higher at every point on the curve. Since only the

fuselage was changed for these studies, the responsibility for the drag increase can be placed wholly on the fuselage.

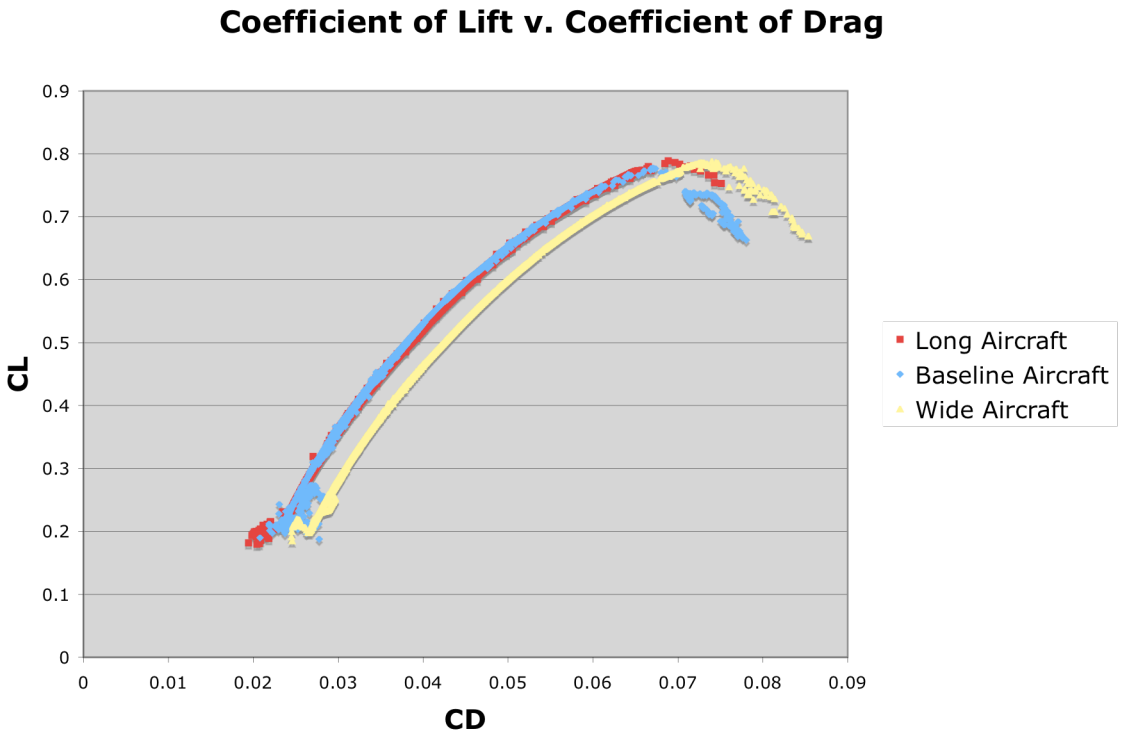


Figure 52: *Drag polars for three test aircraft configurations*

The results for lift/drag as a function of angle of attack (AoA) also reflected the increased drag of the wide configuration, as seen in Figure 53. Here, the slightly higher maximum lift/drag mentioned above is more easily seen.

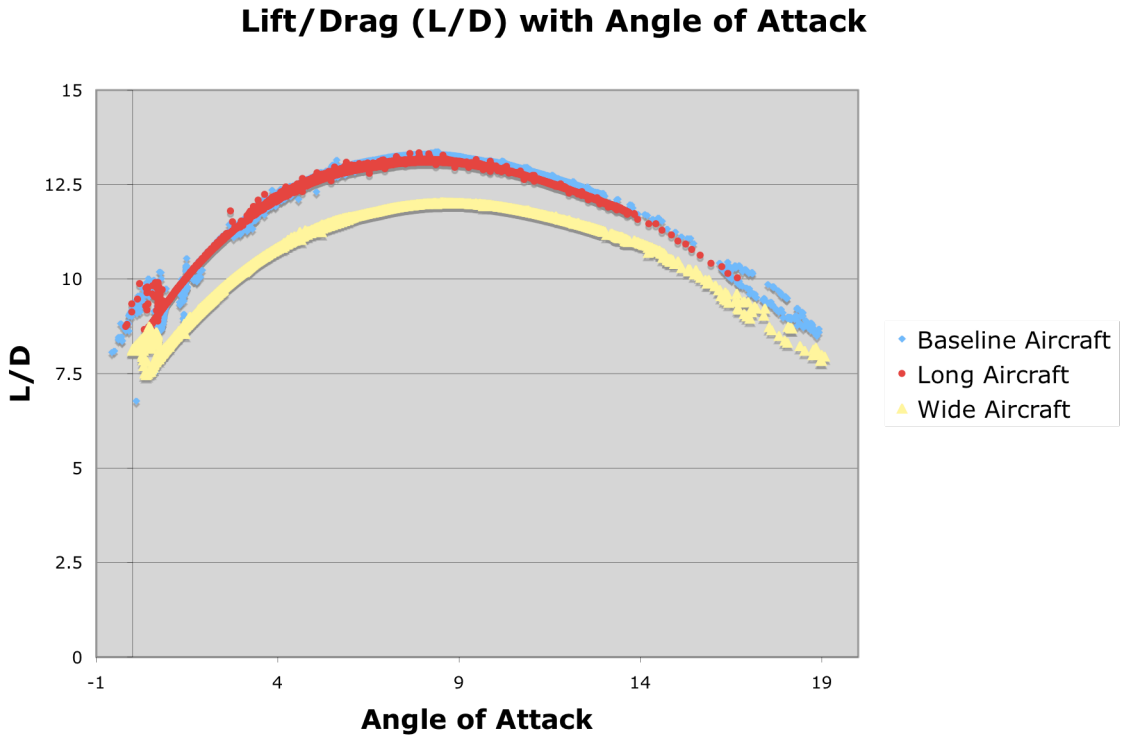


Figure 53: *L/D with AoA for three test aircraft*

8.3 Performance Evaluations

The performance differences that occur as a consequence of the changes in internal layout can now be analyzed rapidly through the *FLOPS* models. The *X-Plane* results described in the last section provided the aerodynamic and engine performance qualities needed by the *FLOPS* model to perform a mission analysis. The three aircraft: baseline, wide, and long, were each flown through the same mission as described above. An additional "high" mission, requiring a cruise and loiter at 8,000 feet was also added to test the limits of the UAVs'

performance. The consequences of various changes on selected performance variables are shown in Table 12.

Table 12: FLOPS *mission performance for three sample aircraft*

	Low Mission Endurance	High Mission Endurance	Loiter Speed	Loiter Power	Maximum Altitude
Baseline	1965	655	41	22%	6923
Wide	1695	634	41	27%	6638
Long	1854	644	42	24%	6782
	minutes	minutes	knots		feet

The results here were also as expected: the baseline aircraft, being smallest and having the best drag performance, also had the best mission performance. At the low altitude, the baseline aircraft was capable of loitering over the target for nearly 33 hours. High altitude performance was not as impressive, however, with the aircraft only managing 11 hours on the high altitude mission. Loiter speed (recorded at the end of the loiter phase, when the aircraft were lightest) was slowest by a small margin, and power used at that point is significantly smaller than the other two aircraft. A final measure of the baseline aircraft's flight superiority was its ceiling. None of the tested aircraft was able to attain the 8,000ft altitude required in the mission profile, but all three aircraft tried, climbing throughout the early phases of the mission as fuel burned away. The baseline aircraft was able to climb higher than the others, reaching an altitude of 6923ft by the end of the loiter phase.

As explored earlier, the long aircraft had a similar drag profile to the baseline aircraft. The absolute drag values were generally slightly higher for the long case than the baseline, but one other effect was the particular shape of the drag curve. The long aircraft flew slightly faster than the baseline due to the slightly different shape of the drag curve, which placed it at a different point on the efficiency curve between the airframe and powerplant. Loiter speed with the long aircraft was a little more than a knot faster, with approximately 10% more power applied to maintain that speed. The drag curve of the wide aircraft, while shifted significantly from the baseline's due to the wide aircraft's increased frontal area, was essentially the same shape. The wide aircraft flew the loiter phase slightly slower than the baseline aircraft, but it required over 20% more power to do so.

The loiter performance reflected an interesting aspect of the UAVs' performance. While the difference in loiter times at the low altitudes was nearly 14% between the baseline and the wide aircraft, with the long version in between, the difference between the high altitude loiter times was less than four percent. An explanation was provided by the power profiles in both cases. The low altitude mission was easily accomplished by all three aircraft, and the power required to sustain flight was far below the maximum that the engines could produce, so the engines were allowed to operate in a more efficient part of their powerbands, resulting in extended loiter times. For the high altitude mission, the continued attempt to climb to the mission altitude and lower power output of the naturally-aspirated engine at high altitudes both contributed to the poor fuel economy at those flight conditions as all three aircraft flew their missions at full

power until the descent. Without the efficient (and long) loiter phase, the baseline aircraft's aerodynamic superiority was minimized.

8.4 Cost

An important contribution to decisions made by the aircraft designer will come in the form of cost. In this case, a simple cost model was developed to analyze potential changes made to the fuselage in order to increase mission functionality. The first task necessary was to determine the construction type used in the production of the aircraft.

The materials used in the Fuel Cell Demonstrator that formed the basis for the weights used in the aircraft analysis are a relatively straightforward composite lay-up, originally utilizing the "disintegrating mold" technique that begins with a lay-up over an original foam mold that is carved away from the final shell[165]. For the production of a commercially available aircraft this would be an undesirable technique because of the time involved creating a new foam mold for every fuselage. While the materials used in the construction of the fuselage would remain the same, a reusable mold would speed the production of aircraft components while simultaneously improving production variability. This mold/lay-up technique would be used to build the fuselage for the notional aircraft under study.

The cost analysis initially centered on a small spreadsheet tool created in *Excel* that was based on a process-based model. This model was designed to

emulate some of the manufacturing data that would be provided by a more thorough cost analysis done as part of an integrated sizing process. This tool used a simple process and material buildup to create cost figures for all three fuselage styles: baseline, long, and wide. After some manipulation, the tool revealed that a simpler cost relationship would be sufficient for this analysis. For an aircraft of this size, the cost of the mold used in production will vary little between the three fuselage choices as the sizes involved are too small to have a large effect on the mold costs. Similarly, material and labor costs were both driven by the same factor – the surface area of the fuselage. For material costs, this is an obvious driver. Composite materials are often priced by the area of material used, so fuselage volume and length will not be a factor in most fuselage designs in this size range.

Labor costs will also scale with fuselage area as the lay-up and finishing of the material is the primary labor cost involved in the process. These two functions are also related to the amount of material, and thus to the fuselage area. The only remaining production cost factors involve use of autoclaves or other automated processing equipment that will not vary in cost between these relatively small changes.

The end result of this analysis was an extremely straightforward cost relationship. Cost of the fuselage changes could all be related directly to the increase or decrease in the surface area of the fuselage under consideration.

8.4.1 Long Versus Wide Fuselage

While the performance of the two fuselage types is significantly different, the costs of producing the pair are similar. The long fuselage features 31% more

surface area than the baseline fuselage, as calculated by the GTS *Transform* utility[166]. The wide fuselage has 32% more area than the baseline and less than a percent more than the long fuselage, so production costs for the pair would be nearly identical. In both cases, the fuselage size was within that possible by two standard 36in widths of composite fabric.

8.4.2 New Versus Derivative Fuselage

With a completely fresh system, the designer is considering which variant of the design to pursue in the later phases. For these purposes, only a relative cost value is required. The relative cost can then be balanced against the possible increase in mission functionality and customer sales associated with the change in fuselage configuration.

In the case of a derivative fuselage design, the baseline aircraft is considered to already be in production. If a customer requires more volume for a larger payload, existing volumetric data from the initial phases of design can be used to supplement data from later phases of the design process, even extending into the operational phases of the aircraft's life-cycle.

The decision here, then, is whether or not the production of more than one airframe variant is worth the investment in new tooling and additional materials. For a single-line production, where a new fuselage would replace the baseline, the resulting aircraft would be larger than required for the baseline cargo, but it would be capable of carrying significantly larger cargo loads than the baseline airframe.

If produced in tandem with the original airframe, the increase in sales due to the increased cargo-carrying capacity would need to account for additional

costs associated with the additional line. A more sophisticated cost analysis is required to fully explore this possibility, as it also requires a detailed market analysis as well as knowledge of existing production rates – neither of which are within the scope of this work. The possibility of such a study does, however, demonstrate the usefulness of volumetric sizing in later phases of the design process.

8.5 Decisions

The ability to assess the volumetric capabilities in this aircraft design has allowed a rapid evaluation of several possible design options. With a design that initially has insufficient space for all of its internal components when constraints are considered, likely solutions involve either redesigning the vehicle or reducing its payload capacity. With traditional design techniques, the discovery that the vehicle had insufficient space would come later in the process, requiring a more expensive redesign. In this test problem, the discovery has been moved to the conceptual phase of the design process, where changes are relatively inexpensive. The early volumetric results also allow customer feedback to more effectively shape the future aircraft.

In this demonstration, the possible options involved either widening the aircraft or lengthening it in order to increase the volume available to internal components. For the design mission, lengthening the fuselage is a superior option for smaller changes in customer cargo, as it makes only modest decreases

in the aircraft's mission capabilities while minimizing the increase in fuselage production costs. For the high-altitude off-design case, the choice is not as significant, as the difference in performance is quite small. In that case, secondary concerns, such as cost of construction, might overwhelm the small difference in flight performance, producing an optimum design that may not be what the designers had originally intended. With customer cargo at the largest size considered, only the wide fuselage meets the fit requirement, producing a small increase in cost but a significant penalty in loiter time.

8.6 Integrating volume with predictive models

The completed *FLOPS* models allow an enhanced look at the design space. While *FLOPS* is not capable of adapting to changes in fuselage volume without assistance, the modified aerodynamic data from *X-Plane* based on the two additional fuselage shapes allows the *FLOPS* model to reflect changes in fuselage shape. Using techniques now common in modern aircraft design, a small analytical model can be created that incorporates volumetric information into a straightforward decision-making aid. In this case, a small set of designs of experiments (DoE) was created around the three aircraft types.

The test design was a simple design of experiments (DoE) using payload and fuel quantity as continuous variables for each of the three aircraft types. With fixed dimensions for each fuselage type recorded as discrete variables and several center and extra points, the resulting DoE was 53 runs. These runs recorded the loiter time for each case as the only response, and a set of

approximations were fit to the results using a standard least squares approach. The results could then be placed in a simple *Excel* tool or used as-is to allow decision-makers to accurately predict the performance of all three aircraft with different payload/fuel loads. The actual versus predicted and residual plots for loiter time (in minutes) are presented in Figure 54 a and b.

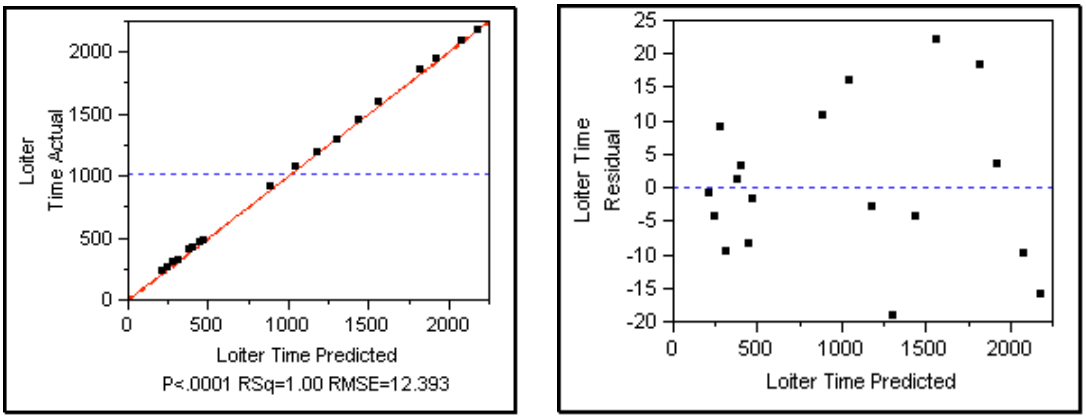


Figure 54: *a) Actual versus predicted and b) Residual plots*

As the figures demonstrate, the results of the model fit are quite good. As the model is fitting a simulation, most of the traditional sources of random error in flight test data are eliminated. A further exploration of the design space, including the modeling of fuselage intermediate shapes, would allow the possibility of creating a model that can rapidly predict the aerodynamic effects of a range of fuselage shapes. Such a study is beyond the scope of this work.

Predicting specific component fit with a metamodel is more difficult. Integrating constraints and irregular shapes with a simple model often yields poor results. Intersections are nonlinear in many cases, reducing the effectiveness of many predictive tools that require a smooth design space. Similarly, using

volume remaining in a constrained design as a guide to surplus space can be ineffective. A design that appears sparse to the casual observer may be on the edge of one or more constraint violations – with little or no space available for additional components.

8.7 Robustness of the final design

In the case of the default fuselage and the results demonstrated above, the customer cargo was a "tight" fit at twice the original design volume, yet a simple subtraction shows that the fuselage still had more than 40% of its total space empty when the cargo was placed. Much of this was due to the loose fit around the fuel tank, but a significant amount of space was empty between components. Even though the customer cargo was close to its absolute size limit within the default fuselage, this still revealed a significant amount of space available for other components.

While the volumetric evaluation of the fuselage with the expanded customer cargo load initiated a study of the design's robustness, inserting uncertainty into the other components allowed a further exploration of the robustness of the baseline design. Increasing the size all of the components except the customer cargo and the fuel tank by ten percent in each dimension results in a volume increase for each component of 33%. That reduces the amount of surplus space in the default fuselage slightly, to around 38%, yet still allows the components to fit properly. Increasing the size of the components

(again keeping the customer cargo and fuel tank constant) results in a total amount of empty space of just over 35%, yet the components cannot fit at this size. This is shown in Table 13, below. A more detailed examination of the fit possibilities shows that increasing the volume of any of the component groups by 73% (an increase in 20% per direction over the baseline) results in a failure to fit properly.

Another interesting feature discovered when exploring component overlaps was the small size of the intersection volumes. Even with the volume of the components nearly doubled, the amount of overlap amounts to less than ten percent of the volume of the smallest component. When the GPS antennas, for example, were increased to the maximum volume while all the other components remained constant, the resulting overlap was less than $0.06in^3$.

Table 13: *Effects of component volume uncertainty*

	Baseline	33% Volume Increase	73% Volume Increase
Cameras	88.1	117.3	152.3
AutoPilot	38.1	50.8	65.9
Transceivers	87.7	116.7	151.5
GPS Antennas	42.6	56.7	73.6
Surplus Volume	40%	38%	35%
Intersection/ Constraint Violation	0	0	1.74
	All Measurements in Cubic Inches		

For unconstrained designs, or for designs with a large number of small components that can be flexibly placed while avoiding constraint boundaries, a

denser design may be simpler to achieve. For the type of design in this study, the combination of constraints, fuselage shape, and otherwise unusable space produces a relatively sparse design despite aggressive packing. The result also demonstrates that the baseline design could accommodate an unexpected increase of 33% in all of the individual component volumes and still converge with the "double-sized" customer cargo. That allows designers a measure of breathing room to allow for unexpected increases in component size – producing a design that is relatively robust to uncertainties in the on-board components. Increases in volume of more than that, however, are not likely to fit, so the robustness does not extend to extreme changes in component size.

9 Conclusions

In a little more than 100 years, powered flight has gone from a curiosity to an everyday occurrence. In this time, the goal of the aircraft designer has changed from attaining the simple miracle of flight itself, through mission performance and financial viability, and into a mature, complex, multicriteria task weaving together a host of disciplines into an aircraft that can perform a host of tasks safely, efficiently, and affordably. As the tasks of the aircraft designer have evolved, so have the tools available and the techniques appropriate for the task. How can simple textbook equations be sufficient when designers have access to computational and analytical tools of ever-increasing power? As tools have evolved for the core disciplines, they have migrated elsewhere as well. Sophisticated simulations can now model in moments a complex integrated aircraft system that once required an expensive prototype.

Volume and internal layout are categories of aircraft design that have, with few exceptions, been handled implicitly at the earliest phases of the design process. Without a technique capable of handling the uncertainty inherent in revolutionary concepts and densely packed conventional designs, that was the only choice. The work presented here has shown that volume and layout information can be incorporated in the conceptual phase of the aircraft design process. *It can intelligently reduce design uncertainty and improve the quality of the final design by bringing internal layout information forward in the design process.*

The ability to handle uncertainty in the internal layout of a design this early in the process also gives designers a chance to eliminate designs that will

not meet all of the customers' desires due to a lack of cargo, fuel, or equipment space. It also gives the designers the ability to account for surplus space in a useful way, rather than having it show up unused at the end of the process. Designers even have the option to make trades in the early phases of design – working with a customer to change a design while being aware of the consequences of design decisions earlier than before.

While the software tool produced for this research to demonstrate a volumetric sizing methodology is a rapid, first-order conceptual tool, not a complete sizing system ready for commercial release, it has shown the effectiveness of the methodology and demonstrated the basic tenets explored in Chapter 2: *Rapid Calculation, Flexible Abilities, Accuracy, and Uncertainty*. The GTS foundation of the code has produced a rapid, robust analysis capability that can handle intersections and volume calculations with the caveats mentioned in Appendix B. The Moreau operator, a hybrid combination of a path-building optimizer and genetic algorithm, has demonstrated significant performance advantages over using either type of optimization scheme alone. The framework created for this research has been shown to work on a range of problems, including a manufacturing cost problem and a UAV design. Results from the code have been checked analytically when possible and qualitatively in more complex problems. Also, the capability to handle uncertainty has been demonstrated in both the manufacturing problem and the UAV design.

9.1 Answering the Research Questions

Any task involving placement of internal components, whether they be in the trunk of a car, the hull of a ship, or the wing of an aircraft, is going to be computationally challenging. The simple nature of object placement involves an extremely large combinatorial space. Even more challenging, this space is not one that is well suited to simple, analytical solutions, instead requiring a solution technique that can avoid large portions of the possibility space without degrading the solution. An effective volumetric sizing method also requires computational efficiency and a good deal of raw computing power. It is, however, possible with modern equipment and intelligent approaches to optimization. With some measure of development, volumetric sizing can be made into an important contributor to the aircraft design process. This reflects onto the research questions posed earlier:

Can volume-based information be included early in the conceptual design phase of a system design?

The work presented here has been created as a standalone process that can be applied at nearly any point in the design process, allowing designers to improve knowledge across a broad swath of the design process. Complete integration of volumetric sizing in a full design environment would allow the consideration of volumetric concerns throughout the design process.

The demonstration problem presented in Chapter 8 completes an initial aircraft conceptual design. Volumetric concerns are explored throughout the

design, resulting in an aircraft visibly and functionally different than if it had been designed without the benefit of volumetric sizing. The resulting aircraft enters the preliminary design phase with a choice of three fuselage configurations depending on customer need, expanding customer input in the early phases of the design process.

Can this information be used to improve decision-making early in the design process?

The most basic benefit to the design process is the ability to filter out designs that do not possess sufficient internal space for all the required components. In addition, designs with surplus space can be redesigned if desired, or the surplus space can be put to use as part of the development process – not wasted at the end. Making these decisions early in the design process reduces costs and potentially increases the performance of the final design.

The three fuselage choices provided in the demonstration problem give the designer the opportunity to address customer concerns early in the design process, where these changes are inexpensive and relatively inexpensive to implement. Traditional techniques would have carried the baseline fuselage configuration to the next design phase without alerting the designer to possible conflicts in customer cargo capacity and internal component fit.

Can this methodology be applied rapidly and easily using readily available computing resources?

While the computational effort needed for internal layout in the conceptual design phase is significant, it is not significantly different than the computational effort needed by other high-fidelity elements of the design process such as computational fluid dynamics studies or complex finite element structural analyses. Further developments in algorithm design, tuning of existing algorithms, and the integration of these techniques into a more polished environment should improve the efficiency of the process and the speed at which results are produced.

The demonstration problem used standard desktop computers running three different operating systems and using two completely different chip architectures. Runtimes for design explorations ranged between one and two days, depending on the fidelity required. The basic code is written in C and portable to most common computing platforms, and the fundamental architecture of the code should be well-suited to operation on high-speed clusters.

Can the results then be passed on to other phases of the design process in such a way to improve the efficiency of the overall design process?

Volumetric sizing offers the ability to share standard shapes throughout the design process. Basic layout information can also be propagated through the design process to assist in later phases of a design's development. Volume checks can also be fed laterally to structure and aerodynamics modules, as well as to controls and performance analyses. The effects of successful integration of

volumetric concerns within a design process can be applied throughout that process.

While the demonstration problem uses only the most basic metadata for transfer of information between phases of the design process, this data and the component position information can be shared between design tools. Currently, the *Blender* visualization software used for creating many of the figures for this work uses a simple file format change in order to properly visualize components in their proper locations. Future applications of volumetric sizing can make use of recent strides in PLM tools used within CAD applications to pass component geometry, constraints, and other qualities among phases of the design process. The basic notion of uncertainty in component size fits neatly within the realm of parametric CAD, again using modern tools to enhance the methodology at the heart of volumetric sizing.

9.2 Addressing the lemmata

Chapter 3 introduced four basic lemmata, smaller supporting hypotheses, into the basic scheme of the research:

1) **Components in a container can be approximated to allow a more rapid exploration of the design space.**

2) **These components can be assigned a degree of uncertainty to account for changes in size and shape with improved levels of knowledge.**

3) **A relatively robust and efficient system can be developed to check components for fit within the boundaries imposed by their system.**

4) **This system will require less time and effort to run than a traditional CAD layout of every component and should be acceptable for the conceptual phase of system design.**

The first lemma was addressed through both the application of simple arithmetic calculations and the creation of more sophisticated surface models for individual components. The simple arithmetic calculations took place off-line, as the total volume required by the internal components was compared to the total available. This simple calculation formed a lower bound on the amount of volume required by the components. The more sophisticated surfaces were created in an attempt to replicate components that would be found in an aircraft of the type analyzed here. The surfaces were created using the GTS library as a set of connected triangles. No internal qualities were considered for the surfaces, though some extra data was preserved in metadata files associated with each

surface in a preview of the potential of future data communication with later phases of the design process.

Uncertainty in the individual components was analyzed through an exploration of the consequences changing the size of individual components would have on the design as a whole. A simple exploration of the "required" internal components was performed as part of an investigation of the baseline fuselage. A more detailed study of customer cargo uncertainty was performed in order to explore the effects of expanding the fuselage to accommodate the extra cargo.

The GTS library was used as the basis of a custom application designed to handle intersections, constraints, and optimization within the context of a volumetric sizing methodology. This application additionally considered inclusions in order to keep components from being placed inside one another. A pair of placement techniques helped ensure rapid and efficient placement of components within the desired fuselage space with as much accuracy as possible. An intelligent optimizer took advantage of the best qualities of gradient-based and domain-spanning optimizers to overcome the limitations of both.

The resulting tool operated rapidly, exploring thousands of possible layout combinations every hour on common computing hardware. It also operated with little human intervention, freeing CAD operators and/or aircraft designers for other tasks. The level of detail in the results was sufficient for use as an initial estimate for internal layout in the preliminary design phase. This result also opened the door to more sophisticated future tools that more completely integrate CAD/PLM tools in volumetric sizing to more comprehensively analyze future designs.

9.3 Future Work

This work should encourage additional research integrating volumetric concerns earlier in the design process in a host of fields. A completely integrated volumetric sizing tool would aid other sizing components in shaping the fuselage and other aircraft components in order to more fully explore the aerodynamic consequences of increasing or decreasing the size of the aircraft. Integration with structural concerns would also be of great benefit, as stringers, ribs, and other supports could be placed in the context of an aircraft's other internal components. Similarly, the structural properties of these components could be used to enhance the properties of the aircraft itself. In the same manner that some motorcycles, for example, use their engines as a structural member, aircraft could make similar use of engines and other internal components that do not currently contribute to the structural integrity of the aircraft as a whole.

More specifically, opportunity exists in enhancing the process of volumetric sizing. A detailed exploration of the impact of resolution and component definition on solution speed could help dramatically speed convergence of volume-related tools. An exploration of different optimization schemes would also help improve the results from these tools.

One of the ultimate goals for volumetric sizing is to assist in the creation of scaling relationships for a host of vehicles. In the general approach presented here, that vision is only a distant possibility, but the complete integration of

volume within the context of the design process can lead to such relations for groups of vehicles much as common empirical relations are used today.

Appendix A: Code Development

This work is reliant on significant application development. At the beginning of the practical work on this effort, *BRL-CAD* had been selected as the primary environment to support all of the necessary intersection and volume-checking requirements. At that point, AML and *PACELAB* were not considered capable of dealing with all of the required features needed for the research. *BRL-CAD* can be scripted, and the initial idea was to script the optimizers into *BRL-CAD* using the TCL scripting language[167].

When this work was formally proposed, the risks inherent in a CAD solution were revealed, and the search was reinitiated for a platform for development of the necessary software. When GTS was selected, this also promoted C as the primary programming language. GTS is a C library, and keeping the entire development environment in C would simplify development somewhat, as it would eliminate concerns over the interface between the C library and, say, a Java main program or a system integrator such as *ModelCenter* or *iSIGHT*.

Unfortunately, this meant that code development would have to start almost entirely from scratch. To this end, a series of working pieces of code were developed on the way to the final version used for the bulk of the work presented here. They are listed below in a (mostly) chronological order.

A1 Preliminary Codes

Moveit (versions 1-3)

Moveit was the first piece of code that made use of basic GTS functions. In its latest iterations, it was able to read in surfaces from files, move them, assess inclusions, measure intersections, and output the results. The functions *make_default*, *inside*, *check_intersect*, and *get_a_point* all were created for this basic code. This is also where the fundamental ideas behind the final code were initiated. Moveit 3 was just over 500 lines long, though it never worked completely as expected.

MemoryLeakHunt

This small application was designed to help locate the memory leaks that plagued initial pieces of code used in this research. GTS takes care of much of its own memory management in conjunction with the GLIB libraries, and it enforces a strict plan of destroying all GTS objects when they are not being used. Not following the correct process can result in memory leaks that will crash any application in a matter of minutes. This application was the first (though not the last) attempt to deal with this issue. MemoryLeakHunt was a little over 200 lines, and was essentially a hard-coded version of Moveit that removed all nonessential code in an attempt to track down the leaks.

MoveitFrame

Another minimalized subset of the Moveit code, this was the first version to properly execute the *get_a_point* function by using the *check_points* function to grab a point from a surface using a remarkably subtle GTS function. This was also the first appearance of the *box_diagonal* function, which helps guide

placement limits for the components. It was over 450 lines and worked well, even if it was of limited functionality.

Gas Tank

The Gas Tank application was initially designed to be a demonstration of a fuel tank-fitting portion of the final code. When it was revealed that this problem is already being pursued by another party, this portion of the research was suspended[xix]. The code as it stands, however, can expand a notional fuel tank to fill a space at any reasonable level of resolution. The tank can expand through holes, though it is relatively stiff, expanding through holes much as a thick balloon would. The code is only a little over 200 lines long as it was designed to be only a component of a larger code, but it produces fascinating results, one of which is shown below in Figure 55. This figure, shown as a hollow black wireframe drawing, is filled with a gray "fuel tank" expanded from a small sphere in the center of the space. The spike in the upper-right of the image shows the fuel tank expanding through a small hole into an additional space. Further modifications to the code produced results that filled most of the small sphere shown there.

[xix] Unfortunately, this information was revealed at a meeting that was covered by a non-disclosure agreement, so no details can be revealed. The author did have further discussions with these parties, however, to confirm that the work presented here is different from the work they were (and still are) pursuing.

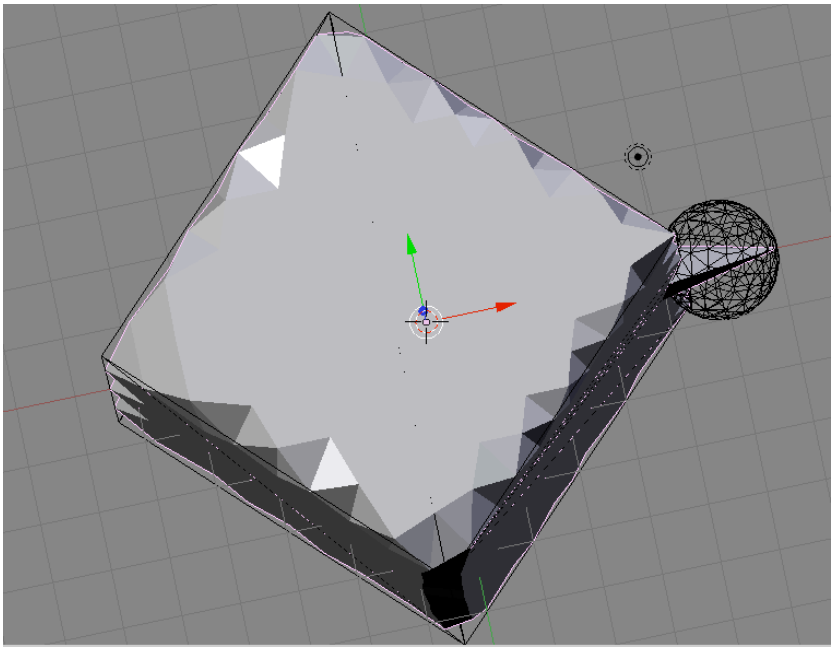


Figure 55: *A complex space filled with a notional fuel tank*

Standalone_Moveit

This code began as *Moveit4*, a simple revision of the initial effort, but evolved into something quite different. Because of the horrible memory leak problems present in early versions of the code, only a very small number of cases could be run. The idea behind *Standalone_Moveit* was to split the code into two separate components. The first, *Standalone_Generator*, would make all the initial calculations required for setup of the problem and then output the sets of coordinates and metadata files to several instances of the *Standalone_Moveit* code, which would then process the information and return the data to the *Standalone_Generator* in order to have the final results processed. There was also a third portion of the code designed to calculate constraint values, *TestConstraints*, but it was never fully developed. Development of this code was suspended when

most of the memory leaks were discovered and eliminated, freeing resources to work on the core of the research rather than on a distributed computing effort. The two components combined for nearly 1,000 lines of code, but that amount would have been much higher had the development not been stopped.

Moveit_OneRun

In many ways, this was the first successful application to come out of this research, and it reached over 700 lines in its final iteration. Loosely based on the earlier *Moveit* codes, *Moveit_OneRun* was significantly different in its fundamental path to a solution. Rather than repeatedly running sets of coordinates in batches, *Moveit_OneRun* simply placed the components one at a time until they fit. This functionality was a simple extension of the basic intelligence required to guarantee component placement in the fuselage. For each component placed, *Moveit_OneRun* simply checked the placement against the fuselage and all of the preceding components. While this worked extremely well for the initial sample problem, it could not be applied in conditions with sensitive constraints. Further, as it had no way to "start over", it was at the mercy of a poor choice of initial position. If the first component was not put in just the right place, it might prevent the problem from converging – even if a solution would be possible otherwise. For these reasons, *Moveit_OneRun* was not pursed for the later portions of the research. The basic mechanism used in this code, however, could form the basis for a different optimization scheme than the one presented here.

Moveit5

Moveit5 was the final iteration of the basic *Moveit* software. Going through more than two dozen significant revisions, its final version generated output in

cases before the creation of the genetic algorithm and its associated framework. At over 800 lines, *Moveit5* was a complete and useful application.

GAMoveit_Test

This was the first attempt to integrate the functions of *Moveit* with the genetic algorithm. It was never completely functional and was soon replaced with *GA_Test_Again*, the final code.

Support Applets

There were several small applets created to solve specific problems that were secondary to the primary code. Here are several of them:

InsideOutside. Measured the percentage of objects placed outside of the fuselage vs the percentage inside for the random placement techniques. It confirmed the analytical result.

TestSimplex. This tested what eventually became the Moreau operator. Unfortunately, the simplex method used in this code was extremely awkward to implement and required the addition of the GSL library, which added yet more complexity that was not worth the trouble. Though similar conceptually to a simplex technique, the final version of the Moreau operator was created by hand.

TestMoreau. This was designed to test the Moreau operator with "live" components and intersections much like those used in the final code. It initially implemented the simplex technique from GSL, but as it never worked properly, it was quickly abandonded.

ReCenter. This small application is used to center GTS components at their calculated center of mass. It is a standalone application that is not officially part of the project, but which can be used to put surfaces at the correct locations for the initiation of the "real" code.

TestIO. This was a simple debugging tool used to generate and read input files used by the rest of the project.

TestSA. This was the code developed to test GSLs simulated annealing code. While promising at first, the SA was extremely difficult to implement, and the restrictions enforced by the structured nature of the library made it a poor choice for use with the bulk of the code and the data structure used to keep track of the component coordinates.

CTests, *Array_Replace* and *More_C_Tests*. These were small applets designed to help track down bugs found in some of the matrix indexing in the main code.

A2 Final Code

The final code used for the majority of the work presented here is *GA_Test_Again*. This code was developed from the selection of earlier code snippets and fully operational applications described above. The code itself is nearly 2000 lines long. It is written in C and makes use of the GTS library for its surface definition and manipulation. The GTS library itself requires the GLIB library, which is best known as an underlying component of the GIMP image manipulation software.

Functionally, GA_Test_Again is made up of a selection of functions. These are presented below:

box_diagonal calculates the diagonal length of a surface's bounding box. This is used for move limits.

all_box_volume calculates box volume equivalents for each component and the container; it also outputs the bounding box dimensions.

check_points and *get_a_point* work in concert to grab a point from a surface to help determine if a surface is inside another surface.

inside uses the results of *check_points* and *get_a_point* as well as the surfaces themselves and their location coordinates to determine if a surface is inside another surface. It is used in later functions in conjunction with the results from the *check_intersect* function.

check_intersect determines the intersection volume of two surfaces. It makes extensive use of GTS' functionality for this purpose.

make_random_point is used by the GA to create new coordinate points.

are_different checks two components' coordinates to see if they are perfectly overlapped. This corrects for an error in GTS when identical surfaces at identical positions are intersected.

make_default creates a metadata file for each component (with associated default values) if the main program does not detect an existing file.

volume_check calculates the total intersection volume between the components and the fuselage as well as amongst each other.

small_swap_GA swaps individual coordinate values as part of the GA crossover function.

ok_changes checks to make sure all coordinate values are within acceptable ranges as determined by the metadata files and some constraints.

make_pop_member makes an entire set of coordinates from scratch. This function is called to generate the initial population for the GA and later to fill in any required population members at the end of each generation.

mutate_GA takes an existing population member and changes the value of several of its coordinates. This is one of the traditional GA operators.

crossover_GA performs crossover operations on selected members of the GA population. This is the second of the traditional GA operators.

Moreau_GA changes the coordinates of a population member based on a simple path-building optimizer. This is not a traditional part of a genetic algorithm, but has been added as part of this research.

constraint_check calculates the value of any constraint violations that may have occurred due to a particular selection of coordinates.

Most of the organizational effort within the code is handled by the main routine. The first portions of the program primarily concern input. The code requires a set of GTS surfaces for the components, a surface for the fuselage, and a short input file that tells the code where to find those files. Metadata for the components is optional, but it makes the constraints possible. Metadata can include things like the component mass, move and rotation limits, and other useful information. If the metadata does not exist, the program will create it with default values. A list of the information associated with the metadata files is shown in Table 14.

Table 14: *Surface metadata variables*

Variable	Value	Default	Notes
Name	string	read in	Default is the name of the component surface file
centerMass	GtsPoint	origin	The center of mass of the component
centerVolume	GtsPoint	origin	The calculated center of volume of the component
origPoint	GtsPoint	calculated	A point on the surface for us in inclusion calculations
mass	real	1	The mass of the component
volm	real	calculated	The volume of the component
volInt	real	not used	Volume of intersection (deprecated)
scaleMe	real	not used	Scale factor (deprecated)
xGravity, yGravity, zGravity	real	0	Used for constraint calculations
intersectVol[]	array of real	not used	Used for intersection storage (deprecated)
locationLimit	real	not used	
scaleLimit	real	not used	
rotLimitX, rotLimitY, rotLimitZ	integer	180	Rotation limits in degrees
rotX, rotY, rotZ	integer	0	Rotations in degrees (deprecated)

Once the fuselage surface is read in, the program then moves the fuselage to its geometric center. It initially assumes that the fuselage is centered at its center of mass, and this position information is retained when the fuselage is moved. The movement reduces the size of the bounding box required to hold the

fuselage, which effectively increases the resolution of the problem without adding any actual combinatorial effort. Components are also assumed to be centered at their centers of mass, but they are also assumed to be placed at the origin. When they are read in, the components are not moved to recenter them, but a small applet is provided to center them at their geometric centers if needed. This applet operates outside of the main application.

The work of note begins after the components are read in. Sets of coordinate points are generated for what will become the initial generation within the GA. Depending on user input at compile-time, the population generation can be done completely at random, with the risk that components will not be placed in the fuselage or with "intelligence". This basic intelligence will guarantee that each component is completely inside the fuselage by checking for inclusion and intersection at each component placement. This will significantly increase the time required to run the application, but it also significantly improves the convergence rate. Evidence suggests that the use of "intelligence" for the population generation is normally a good idea, though tightly-packed spaces can operate better with the random placement. Once the first generation is created, it is assessed. Intersection volume and constraints are both checked, and each population member is assigned values accordingly.

The beginning of the second generation also signifies the true beginning of the genetic algorithm. This begins with the reproduction phase. In this case, rather than a tournament selection or some other semi-stochasatic reproduction tool, a simple list of the best members is produced instead. Since this list can be useful later in the optimization, its cost in computational effort is considered worth the results.

Once the best population members are selected, the remainder is sifted through for mutation. Random population members are selected and then have elements of their coordinate structure changed at random. This does allow components to be placed outside the fuselage structure, but components are still limited to the sphere that contains the fuselage's bounding box. These mutated cases make up the next user-defined group of the population. Population members who had components outside the fuselage initially will often produce extremely bad results after mutation, so the intelligent population selection is again recommended.

The Moreau operator normally functions next. Coordinates of population members chosen for "treatment" are perturbed slightly in a simple path-building optimizer that "nudges" them a user-defined distance. Rotations are handled similarly, with initial limits set by the user at compile time. In order to maximize the effectiveness of these "nudges", the "best" cases from the prior generation are chosen for "treatment". In the event that more Moreau cases are called for than the number of preserved cases from the prior generation, the Moreau operator then randomly selects from the rest of the population. As an alternative, with a small modification to the code, the Moreau operator can be made to run on every case. This adds tremendously to the processing time for each generation but may be useful for certain arrangements. Late runs in the final problem were run outside of the GA in this manner, but only on selected cases.

After the Moreau operator is complete, crossover begins. Pairs of randomly selected members of the population have several coordinates swapped. The coordinates to be swapped are also chosen at random. The

number and starting position of the coordinates are chosen in addition to which coordinates (x,y,z or ax,ay,az) will be swapped.

Once the crossover operation is complete, any remaining positions are filled by the same random process used for the initial population (and choices about process intelligence are preserved). The process then begins again, with the selection of the "best" population members. The entire process is repeated as many times as desired, though limits have been added to stop the process earlier if desired.

Appendix B: Computing Resources

The majority of code development for this project was centered on two Macintosh G5 computers. Code was written in Apple Corporation's XCode development environment. All code was written in standard C and compiled using the GCC compilers used as part of XCode. Code run remotely was run in Linux and compiled locally, again using GCC compilers. Most development testing was done on CFD2, Orville and Wilbur, Intel-based computers within ASDL. Compile-time optimization was kept to a minimum until later work in an attempt to maintain maximum compatibility.

The GTS library was updated during development from version 0.7.3 to 0.7.6. This made no observable difference in performance or results.

The GLIB library used by GTS was also updated during development. The move to version 2.9.12 reduced memory leaks significantly, allowing more runs before the leaks caused problems. Installing and linking to the correct version of the GLIB libraries was extremely difficult on Wilbur and the other remote machines.

B1 Hardware

Primary development was done on two Macintosh G5 computers. The first, a 1.6Ghz Power Macintosh G5, is located within ASDL. The second, a 2Ghz iMac G5 is located at the author's home.

The Linux machines were all located within ASDL or at a remote location on the Georgia Tech campus monitored by the Office of Information Technology. The primary machines used here were CFD2, Orville and Wilbur, Intel-based computers running a version of Red Hat Linux.

Work with SAS Institute's JMP software was performed on a Sony Vaio equipped with an Intel Pentium 3.

Details concerning the computers used in this work are provided in Table 15.

Table 15: *Development hardware*

Name	Processor	Speed	RAM	Disk Storage	Operating System
iMac G5	IBM PowerPC G5	2.0Ghz	1.5GB	250GB	Mac OSX 10.4.9
Power Macintosh G5	IBM PowerPC G5	1.6Ghz	1.0GB	160GB	Mac OSX 10.4.9
CFD2	Intel Pentium 4	3.2Ghz	1.0GB		Fedora Core
Orville	Intel Xeon	3.2Ghz	4.0GB		Red Hat Enterprise 4
Wilbur	Intel Xeon	3.2Ghz	4.0GB		Red Hat Enterprise 4

B2 Software

The primary development environment for this project was Xcode. This software is included with Apple Inc.'s Developer Tools, a free download at the time of this writing[168]. Xcode is a fully-featured integrated development environment that is primarily for application development on the Macintosh Operating System. Fortunately, Xcode provides solid support for legacy code, and it is capable of supporting standard versions of the C programming language. Xcode uses the standard GCC compilers for programs written in C. This provides excellent compatibility with the GCC compilers on other platforms.

Since Xcode contains all of the necessary background code to generate its own automated build tools, there is no need for a separate makefile for compilation in OSX. For work in Linux, a makefile was required. This was created by Rob MacDonald and customized for the GTS environment already installed on the CFD2 computer.

B2.1 GTS and other Applications: Capabilities and Issues

As mentioned earlier, GTS was used as the primary tool for handling component geometry in this work. A typical installation of GTS also includes several standalone tools used as utilities. The *gts2stl* and *stl2gts* utilities were commonly used for switching back and forth between the GTS environment and the .stl files used by the *Blender* visualization software. *Blender* was used only as a visualization tool in this work, though that is not its intended role[169]. The GTS

transform utility was used for scaling and to calculate information about individual components.

Outside of difficulties learning the particular qualities of GTS, it was a reasonably stable platform for surface generation and manipulation. Specific issues revolved around three particular problems. The first was the conversions between GTS files and the STL files used by *Blender*. As mentioned earlier, both GTS and Blender will accept files that have inconsistent notions of "inside" and "outside"; unfortunately, the tools needed to rectify that issue are unpredictable, and a good deal of time was spent tweaking shapes that should not have been difficult to manipulate.

The next problem was the abovementioned memory leaks. Related to GTS' communication with the GLIB library, the memory leaks were not entirely remedied during the course of this research. The size of the problems explored here was partly driven by a need to avoid problems with the memory leaks. As mentioned above, the memory leaks only really became an issue for large runs lasting on the order of 3-4 days. Some slowing was evident before that, but the code typically ran well for two days or so on most of the computers used for this work.

The most serious problem with GTS, however, was its problems with certain Boolean operations. GTS will not properly detect intersections between coplanar line segments. Normally, this only occurs when identical components are compared while offset from each other only in one dimension or if completely collocated. While this is not generally a problem, the large number of operations typical of cases run in this research, and the common reuse of shapes, meant that this situation did, in fact, occur with enough regularity to cause

concern. The long runtimes for code in this work also made the problem hard to track, as the program would run fine for hours before the problem occurred.

The initial solution was to implement a check in the code for collocation. If components were placed at exactly the same coordinates, the volume of intersection was assumed to be the volume of one of the components (true, for identical components); the formal intersection in these cases was never performed. This worked initially, as most of the problems had occurred when identical components were placed in the same position with the same rotation values. For components that were merely in the same plane as each other, the simple analytical solution was ineffective, as a straightforward check like the earlier solution was not practical to implement. The solution was to reduce the size of several individual components by approximately three percent (by volume). This has apparently eliminated the problem, as the line segments being compared are now slightly different. The source of the problem within GTS also appears to be through its interface with the GLIB graphics library, so a fix may not be forthcoming from the GTS developers.

B2.2 Support Applications

The *Microsoft Office* suite of tools was also used to prepare this document, supporting presentations, and a host of charts, graphs, and tables[170]. Excel was also used to create the cost model used in Chapter 7 and in support of the X-Plane aerodynamic and engine data used in Chapter 8. Conversions of the resulting *Word* documents to PDF were handled by the PDF converter built in to OSX. Image handling was done primarily by Apple software as well, with scanner support provided though *ImageCapture* and file format conversion done

with the bundle version of Lemke Software's *GraphicConverter*[171172]. SAS Institute's JMP statistical analysis software was used to build and analyze the multiple design of experiments used in support of this work[173].

References

[1] Hill Aerospace Museum. Online Viewed January 2007. Available at http://www.hill.af.mil/museum/photos/coldwar/c-140.htm. Note: the website was redesigned on 23 February, 2007, and some pages are missing.

[2] Photo by the author. USAF Museum in Dayton, OH 2003.

[3] Owen, K. *Concorde, New Shape in the Sky*. Jane's Publishing Company Limited. London. 1982. Pg. 64

[4] Federation of American Scientists. *Joint Strike Fighter (JSF).* Online. Accessed January 2007. Information available at: http://www.fas.org/man/dod-101/sys/ac/jsf.htm

[5] Lockheed Martin. *Lockheed Martin F-35 Lightning II Stealth Fighter Completes First Flight*. Online. Viewed 24 February 2007. Information available at: http://www.lockheedmartin.com/wms/findPage.do?dsp=fec&ci=18077&rsbci=0&fti=112&ti=0&sc=400

[6] Federation of American Scientists. *F-22 Raptor*. Online. Accessed January 2007. Information available at: http://www.fas.org/man/dod-101/sys/ac/f-22.htm

[7] Badrocke, M. and Gunston, B. *Lockheed Aircraft Cutaways. The History of Lockheed Martin*. Osprey Publishing Ltd. London. 1998.

[8] Mavris, D., Soban D., Upton, E., DeKock, B., and Harcrow, A. *System Level Assessment of an Uninhabited Aerial Vehicle Utilizing Fuel Cell Technology*. Aerospace Systems Design Laboratory. Submitted to NAVAIR. Technical Presentation. November 2005.

[9] Ibid.

[10] Wynbrandt, James. *Proofs Positive. Cessna unveils LSA, Next Generation proof of concept aircraft*. Experimental Aviation Association. Online. Viewed January 2007. Available at: http://www.airventure.org/2006/tuesjuly25/cessna.html.

[11] Cessna Aircraft Company. *Skyhawk. Put Fun Into Fundamentals*. Online. Viewed 29 September 2006. Available at http://skyhawk.cessna.com

[12] Photo by Kenneth Kovac/AVWEB. Available at: http://www.avweb.com/newspics/new_cessna2.jpg

[13] Photo by Cessna. Available at: http://skyhawk.cessna.com/gallery.chtml#

[14] Anderson, J.D. *Aircraft Performance and Design*. Boston: McGraw-Hill, 1999. Pg. 21.

[15] Christy, J. *American Aviation, an Illustrated History*. Blue Ridge Summit: TAB Books, 1987. Pg 4.

[16] Ibid. pg 18.

[17] Anderson, J. D. *Fundamentals of Aerodynamics*, Second Edition. New York: McGraw-Hill, 1991. Pg. 324

[18] Ibid. Pg. 553.

[19] Raymer, D. P. *Aircraft Design: A Conceptual Approach*. 3rd ed. Reston: American Institute of Aeronautics and Astronautics, 1999. Pg 4.

[20] Mavris, D., Soban D., Upton, E., White, T., and Mattson, J. *Technology Assessment of a Fuel Cell Auxiliary Power System for Naval Aviation*. Aerospace Systems Design Laboratory. Submitted to NAVAIR. Technical Report. October 2004. Pg 66-67.

[21] Mavris, D., Soban D., Upton, E., White, T., and Mattson, J. *Technology Assessment of a Fuel Cell Auxiliary Power System for Naval Aviation*. Aerospace Systems Design Laboratory. Submitted to NAVAIR. Technical Report. October 2004. Pg 34.

[22] Lindbergh, Charles A. *The Spirit of St. Louis*. New York: Charles Scribner's Sons. 1953. Pg 87.

[23] McDonnell Douglas. *C-17 Globemaster III Technical Description and Planning Guide*. McDonnell Douglas Aerospace Transport Aircraft. Long Beach, CA. 1993. MDC 93K0048. Pg I-1-4

[24] "Aero Spacelines Super Guppy". From *Wikipedia*. Online. Viewed 31 March, 2007. Available at: http://en.wikipedia.org/wiki/Super_Guppy

[25] 747 LCF First Flight announcement available at: http://www.boeing.com/commercial/747family/news/2006/q3/060909a_nr.html

[26] Photo by the author. Dayton, OH. 2003.

[27] Nam, T. et al. *A Generalized Aircraft Sizing Method and Application to Electric Aircraft*. American Institute of Aeronautics and Astronautics. 3rd International Energy Conversion Engineering Conference. August 2005. AIAA 2005-5574.

[28] Kirby, M., Mavris, D. *A Method for Technology Selection Based on Benefit, Available Schedule, and Budget Resources*. Presented at the 2000 World Aviation Conference. San Diego, 2000. AIAA 2000-01-5563. Pg. 4

[29] Raymer Daniel P. *Enhancing Aircraft Conceptual Design using Multidisciplinary Optimization*. Diss. Kungul Tekniska Högskolan, 2002. Stockholm, 2002. Pg 78.

[30] Mavris, D., Baker, A., Schrage, D. *IPPD Through Robust Design Simulation for an Affordable Short Haul Civil Tiltrotor*. Presented at the American Helicopter Society 53rd Annual Forum. Virginia Beach, 1997. Pg 2.

[31] Mavris, D., Baker, A., Schrage, D. *IPPD Through Robust Design Simulation for an Affordable Short Haul Civil Tiltrotor*. Presented at the American Helicopter Society 53rd Annual Forum. Virginia Beach, 1997. Pg 2. Chart is after Fabrycky and Blanchard.

[32] Ibid.

[33] Ibid.

[34] Masson, P. et al. *HTS Motors in Aircraft Propulsion: Design Considerations*. IEEE Transactions on Applied Superconductivity, Vol. 15, no. 2, June 2005. Pg 2221.

[35] Boeing 767 data available from Boeing at http://www.boeing.com/commercial/767family/news.html

[36] B52H data available at http://www.fas.org/nuke/guide/usa/bomber/b-52.htm

[37] Photo courtesy of Boeing. Online. Viewed 9 February 2007. Available at www.boeing.com/commercial/767family/pf/pf_400_pod.html

[38] Photo Courtesy Tony Landis/NASA. Online. Viewed 9 February 2007. Available at http://www.dfrc.nasa.gov/Gallery/photo/B-52/index.html

[39] Raymer Daniel P. *Enhancing Aircraft Conceptual Design using Multidisciplinary Optimization*. Diss. Kungul Tekniska Högskolan, 2002. Stockholm. pg 74.

[40] Schrage, D. P. *Technology for Rotorcraft Affordability Through Integrated Product/Process Development*. Presented at the American Helicopter Society 55th Annual Forum. AHS, 1999. Pg 3.

[41] Ibid.

[42] Roskam, Jan. *Airplane Design*. 8 vols. Ottawa: Roskam Aviation and Engineering Corporation, 1989.

[43] Roskam, Jan. *Airplane Design*. vol 2. Ottawa: Roskam Aviation and Engineering Corporation, 1989. pg 4.

[44] Anemaat, W., Kaushik, B. "Minimize Center of Gravity Range in Aircraft Preliminary Design Using Advanced Aircraft Analysis". Oral Presentation. Society for Automotive Engineers. Witchita, KS. August 2006.

[45] Raymer Daniel P. *Enhancing Aircraft Conceptual Design using Multidisciplinary Optimization*. Diss. Kungul Tekniska Högskolan, 2002. Stockholm, 2002. Pg 74 – 80

[46] Raymer, Daniel P. *Aircraft Design: A Conceptual Approach*. 3rd ed. Reston: American Institute of Aeronautics and Astronautics, 1999. (pg 699-700)

[47] Ibid. Pg 313

[48] NAVSEA. *Advanced Surface Ship Evaluation Tool (ASSET) Introduction Training Course*. Training Manual. Carderock Division Naval Surface Warfare Center Ship Design Tools. 2003.

[49] Howe, Denis. *Aircraft Loading and Structural Layout*. Reston: American Institute of Aeronautics and Astronautics, 2004.

[50] Ibid. Pg. 2

[51] Information about Technosoft's AML is available online at: http://www.technosoft.com/aml.php/

[52] Dahl, Jurgen. Technosoft. AMRaven training program. 24 January 2007.

[53] National Aeronautics and Space Administration. *Flight Optimization System Users Guide,* release 6.0.3 by L.A. McCullers. 13 August 2003. Electronic.

[54] Gelhausen, P. et al. *Overview of ACSYNT for Light Aircraft Design*. SAE Paper 951159. Pg. 1.

[55] Ibid. Pg 8.

[56] Mattingly, J.D., Heiser, W. H., and Daley, D. H.. *Aircraft Engine Design*. Washington: American Institute of Aeronautics and Astronautics, 1987.

[57] Ibid. Pg 22.

[58] National Aeronautics and Space Administration. *Flight Optimization System Users Guide,* release 6.0.3 by L.A. McCullers. 13 August 2003. Electronic.

[59] Van der Velden, Alex. Engeneous Software. Personal Discussion. 5 May, 2005.

[60] Ibid.

[61] Owen, K. *Concorde, New Shape in the Sky*. Jane's Publishing Company Limited. London. 1982. Pg. 72

[62] Badrocke, M. and Gunston, B. *Lockheed Aircraft Cutaways. The History of Lockheed Martin*. Osprey Publishing Ltd. London. 1998.

[63] Anderson, J.D. *Aircraft Performance and Design*. Boston: McGraw-Hill, 1999. Pg 295.

[64] Khoury, Gabriel A. and Gillett, J. David, ed. *Airship Technology*. Cambridge: Cambridge University Press, 2002. Pg 15.

[65] Ibid. Pg 14.

[66] Ibid. Preface, Pg xi.

[67] U-Haul. Chart available at www.uhaul.com. Online, 7 August 2005.

[68] U-Haul. Chart available at www.uhaul.com. Online, 7 August 2005.

[69] U-Haul. www.uhaul.com Online. 7 August 2005

[70] www.google.com. Online search. *cargo loading software*. 8 August 2005. Of course, many of the Google results have nothing to do with the desired software, but the sheer number of valid results was impressive.

[71] www.softtruck.com. Offers loading of rectangular boxes and a graphical axle loading interface.

[72] Tops Engineering MaxLoad software available at www.topseng.com. Allows bottles as well as rectangular boxes.

[73] Load Captain software, by Horizon Services Group, is available at www.loadcaptain.com. Loads boxes, cylinders, drums, and pallets.

[74] Loadplanner is available at www.loadplanner.com.

[75] Gillmer, T. and Johnson, B. *Introduction to Naval Architecture*. United States Naval Institute. Annapolis, 1982. Pg 40.

[76] NAVSEA. *Advanced Surface Ship Evaluation Tool (ASSET) Introduction Training Course*. Training Manual. Carderock Division Naval Surface Warfare Center Ship Design Tools. 2003.

[77] Corcoran, A.L. III and Wainwright, R.L. *A Genetic Algorithm for Packing in Three Dimensions*. ACM 0-89791-502-X/92/0002/1021. 1992.

[78] DeLaurentis, D. *A Probabilistic Approach to Aircraft Design Emphasising Stability and Control Uncertainties*. Georgia Institute of Technology. Diss. November 1998. Pg. 1

[79] Roskam, Jan. *Airplane Design*. vol 2. Ottawa: Roskam Aviation and Engineering Corporation, 1989. pg 104.

[80] Boeing Company. Online. Viewed 23 February 2007. 777 data available at: http://boeing.com/commercial/777family/pf/pf_200product.html.

[81] Demaine, E.D., Hohenberger, S., and Liben-Nowell, D. *Tetris is Hard, Even to Approximate*. Cornell University. Online. arxiv.org/abs/cs.cc/0210020. Pg 3.

[82] Black, Paul E. "knapsack problem", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 26 January 2005. Viewed 17 March 2007. Available at: http://www.nist.gov/dads/HTML/knapsackProblem.html

[83] National Aeronautics and Space Administration. "Survivor on the Moon", part of *the NASA Explores* series of educational materials. Online. Viewed 17 March 2007. Available at: http://media.nasaexplores.com/lessons/01-026/5-8_1.pdf

[84] Cagan, J. Shimada, K., and Yin, S. "A Survey of Computational Approaches to Three-Dimensional Layout Problems". Submitted to Computer-Aided Design, 2000. Online. www.andrew.cmu.edu/user/shimada/papers/00-cad-cagan.pdf. Pg. 11

[85] Li, H. Antonsson, E. "Genetic Algorithms in MEMS Synthesis" In Proceedings of IMECE'98. 1998 ASME International Mechanical Engineering Congress and Expositions. Anaheim, California. November 15-20, 1998.

[86] Cagan, J. Shimada, K., and Yin, S. "A Survey of Computational Approaches to Three-Dimensional Layout Problems". Submitted to Computer-Aided Design, 2000. Online. www.andrew.cmu.edu/user/shimada/papers/00-cad-cagan.pdf. Pg. 1

[87] NAVSEA. *Advanced Surface Ship Evaluation Tool (ASSET) Introduction Training Course*. Training Manual. Carderock Division Naval Surface Warfare Center Ship Design Tools. 2003.

[88] Seiden, Steven S. *On the Online Bin Packing Problem*. Journal of the ACM, Vol. 49, No. 5, September 2002, pp 640-671. ACM 0004-5411/02/0900-0640

[89] Bentley, J.L. et al. *Some Unexpected Behavior Results for Bin Packing*. ACM 0-89791-133-4/84/004/0279

[90] Corcoran, A.L. III and Wainwright, R.L. *A Genetic Algorithm for Packing in Three Dimensions*. ACM 0-89791-502-X/92/0002/1021. 1992.

[91] Weisstein, Eric W. "NP-Problem." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/NP-Problem.html. Online. 26 May, 2005.

[92] Weisstein, Eric W. et al. "Polynomial Time." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/PolynomialTime.html. Online. 26 May, 2005.

[93] Weisstein, Eric W. "NP-Problem." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/NP-Problem.html. Online. 26 May, 2005.

[94] Weisstein, Eric W. "NP-Problem." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/NP-Problem.html. Online. 26 May, 2005.

[95] Cook, William. *Travelling Salesman Problem*. School of Industrial and Systems Engineering at the Georgia Institute of Technology. Online. Viewed 23 February 2007. Available at: http://www.tsp.gatech.edu/index.html

[96] Corcoran, A.L. III and Wainwright, R.L. *A Genetic Algorithm for Packing in Three Dimensions*. ACM 0-89791-502-X/92/0002/1021. 1992.

[97] Ibid.

[98] Clay Mathematics Institute. *P vs. NP Problem*. Online. Viewed 17 March 2007. Available at: http://www.claymath.org/millennium/P_vs_NP/. Information on the prizes in general is available at: http://www.claymath.org/millennium/

[99] CATIA is marketed by IBM and can be found online at: www-306.ibm.com/software/applications/plm/catiav5. Information can also be found from Dassault at www.3ds.com/products-solutions/plm-solutions/catia.

[100] Image from Dassault Systemes. Available at http://www.3ds.com/products-solutions/plm-solutions/catia.

[101] Information and links to downloads available at www.brlcad.org. The Army Research Lab is available at www.arl.army.mil

[102] The image and creation details are available at http://sourceforge.net/project/screenshots.php?group_id=105292.

[103] The image and creation details are available at http://sourceforge.net/project/screenshots.php?group_id=105292.

[104] Information and links to downloads available at www.opencascade.org. OpenCASCADE S.A. is available at www.opencascade.com.

[105] Ibid. The application shown in the figure is Samtech's SamCEF Field software. Information is available at www.samtech.fr

[106] Information and links to downloads available at gts.sourceforge.net.

[107] triAero can be found online at: http://triaero.sourceforge.net/

[108] Stichting Blender Foundation. *Blender* Software. Online. Viewed 23 February 2007. Software available at: http://www.blender.org.

[109] Cagan, J. Shimada, K., and Yin, S. A Survey of Computational Approaches to Three-Dimensional Layout Problems. Submitted to Computer-Aided Design, 2000. Online. www.andrew.cmu.edu/user/shimada/papers/00-cad-cagan.pdf

[110] Ibid.

[111] Wikipedia. *Branch and Bound*. Online Viewed 17 March 2007. Available at: http://en.wikipedia.org/wiki/Branch_and_bound

[112] Gray, P. et al. "Branch and Bound". in *A Survey of Global Optimization*. Sandia National Laboratories. Online. Viewed 17 March 2007. Available at: http://www.cs.sandia.gov/opt/survey/branch-and-bound.html

[113] Ibid

[114] Wells, H.G. "The Island of Dr. Moreau". *Masters Library H. G. Wells*. Minneapolis: Amaranth Press, 1979. Pg 69-157.

[115] Csirik, J. Johnson, D. S. et al. "On the Sum-of-Squares Algorithm for Bin Packing". STOC 2000. Portland Oregon. ACM 2000 1-58113-184-4/00/5

[116] Kenyon, C. "Best-Fit Bin-Packing with Random Order". Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. Ch 41. Philadelphia: Society for Industrial and Applied Mathematics, 1996. Pg 359-364

[117] Ibid.

[118] Kenyon, C. "Best-Fit Bin-Packing with Random Order". Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. Ch 41. Philadelphia: Society for Industrial and Applied Mathematics, 1996. Pg 359-364

[119] Csirik, J. Johnson, D. S. et al. "On the Sum-of-Squares Algorithm for Bin Packing". STOC 2000. Portland Oregon. ACM 2000 1-58113-184-4/00/5

[120] Ibid.

[121] Kenyon, C. "Best-Fit Bin-Packing with Random Order". *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. Ch 41. Philadelphia: Society for Industrial and Applied Mathematics, 1996. Pg 359-364.

[122] Ibid.

[123] Ibid.

[124] Boeing Company. Online. Viewed 23 February 2007. 777 data available at: http://boeing.com/commercial/777family/pf/pf_200product.html.

[125] Nam, T. et al. *A Generalized Aircraft Sizing Method and Application to Electric Aircraft*. American Institute of Aeronautics and Astronautics. 3rd International Energy Conversion Engineering Conference. August 2005. AIAA 2005-5574.

[126] PACE. Online. Viewed 21 March 2007. PACELAB is available from at http://www.pace.de/en/products.php

[127] Technosoft. Online. Viewed 23 February 2007. Information about AML is available at http://www.technosoft.com/.

[128] Dahl, Jurgen. Technosoft. AMRaven training program. Atlanta, Georgia. 24 January 2007.

[129] Information about GTK, GNOME, and GLIB can be found at http://www.gtk.org/

[130] Ibid.

[131] Jang, S.H. Rhee, K.Y. "An Application of Annealing Algorithm to the Layout Design of Ocean Space Submergible Boat", JSME International Journal Series , Vol. 47, No. 2 (2004). Pg 770-775.

[132] The GSL Library can be found at http://www.gnu.org/software/gsl/.

[133] Vanderplaats, Garret N. *Numerical Optimization Techniques for Engineering Design*. 3rd Ed. Colorado Springs: Vanderplaats Research and Development, Inc. 1999. Pg 189-190.

[134] Ibid. Pg 190.

[135] Clausen, Jens. *Branch and Bound Algorithms – Principles and Examples*. Department of Computer Science, University of Copenhagen. 12 March 1999. Viewed 23 February 2007. Available Online at: http://www.imada.sdu.dk/Courses/DM85/TSPtext.pdf. Pg 2.

[136] Black, Paul E. "branch and bound", in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 12 September 2005. Viewed 23 February 2007. Available from: http://www.nist.gov/dads/HTML/branchNbound.html

[137] Pintér, János. "Branch and Bound Algorithm." From Mathworld -A Wolfram Web Resource, created by Eric W. Weisstein. http://mathworld.wolfram.com/BranchandBoundAlgorithm.html

[138] Yin, S. Cagan, J. "Exploring the Effectiveness of Various Patterns in an Extended Pattern Search Layout Algorithm". in *Transactions of the ASME*. Vol. 126 July 2004.

[139] Ibid

[140] Vanderplaats, Garret N. *Numerical Optimization Techniques for Engineering Design*. 3rd Ed. Colorado Springs: Vanderplaats Research and Development, Inc. 1999. Pg. 94-103.

[141] Ibid. Pg 49-52.

[142] Phoenix Integration's Model Center Software can be found at: http://www.phoenix-int.com/products/modelcenter.php

[143] Engineous Software's iSIGHT software can be found at: http://www.engineous.com/product_iSIGHT.htm

[144] Image from Phoenix Integration. http://www.phoenix-int.com/products/modelcenter.php

[145] Information about Technosoft's AML can be found here: http://www.technosoft.com/aml.php/

[146] The Mathworks, Inc. "Graphics: Group Objects". Online. Viewed 1 April, 2007. Available at: http://www.mathworks.com/access/helpdesk/help/techdoc/index.html?/access/helpdesk/help/techdoc/creating_plots/f7-40355.html

[147] Nam, T. et al. *A Generalized Aircraft Sizing Method and Application to Electric Aircraft*. American Institute of Aeronautics and Astronautics. 3rd International Energy Conversion Engineering Conference. August 2005. AIAA 2005-5574.

[148] Stichting Blender Foundation. *Blender* Software. Online. Software available at: http://www.blender.org.

[149] Garcia, E., Marx, W., and Mavris, D. N. ALCCA User Notes. February 1999.

[150] Information on the F-16 is available at http://www.fas.org/man/dod-101/sys/ac/f-16.htm

[151] Wilson, J.R. "2005 AIAA Worldwide UAV Roundup". In *UAVs: a Worldwide Roundup*. American Institute of Aeronautics and Astronautics. Online. Viewed 22 March, 2007. Available at: http://www.aiaa.org/aerospace/Article.cfm?issuetocid=365&ArchiveIssueID=39

[152] McKinley, Robert. Director of Engineering, DRS - Unmanned Technologies. Personal Discussion. 2 March, 2007.

[153] National Aeronautics and Space Administration. *Flight Optimization System Users Guide,* release 6.0.3 by L.A. McCullers. 13 August 2003. Electronic.

[154] Bradley, T. H., Moffitt, B., Thomas, R., Parekh, D. E. and Mavris, D., 2006. *Test Results for a Fuel Cell-Powered Demonstration Aircraft*. in Society of Automotive Engineers Power System Conference, November 7-9, 2006, New Orleans. 2006-01-3092.

[155] Ibid.

[156] Graphic by Danielle Soban.

[157] Stichting Blender Foundation. *Blender* Software. Online. Software available at: http://www.blender.org.

[158] Upton, E. *Mommy, When do I get to Play? Using X-Plane for Real Design*. Internal White Paper. August 2001.

[159] Upton, E. et al. *URETI Task 4: General Aviation Modeling*. Presented to the Georgia Tech Aerospace Systems Design Lab External Advisory Board. May, 2004.

[160] Meyer, Austin. "How it Works". In *Description*. Online. Viewed 25 March, 2007. Available at www.X-Plane.com/about.html

[161] Abbott, I. Von Doenhoff, A. *Theory of Wing Sections*. New York: Dover Publications, Inc. 1959.

[162] Ibid. Pg 111-123.

[163] Ibid.

[164] National Aeronautics and Space Administration. *Flight Optimization System Users Guide,* release 6.0.3 by L.A. McCullers. 13 August 2003. Electronic.

[165] Broughton, Adam. Personal Communication. 21 April, 2007.

[166] Information and links to downloads available at gts.sourceforge.net.

[167] Anderson, J. et al. *BRL-CAD Overview*. Online. Viewed 22 March 2007. Available at: http://www.brlcad.org/overview.html

[168] Information and software downloads available at: http://developer.apple.com/

[169] Stichting Blender Foundation. *Blender* Software. Online. Software available at: http://www.blender.org.

[170] Information available at: http://www.microsoft.com/mac/products/office2004/office2004.aspx?pid=office2004

[171] Information about OSX 10.4 is available at: http://www.apple.com/macosx/tiger/

[172] Lemkesoft GmbH and the GraphicConverter software are available at: http://www.lemkesoft.com/

[173] SAS Institute. *JMP | Software – Data Analysis – Statistics – Six Sigma – DOE*. Online. Viewed 28 April, 2007. Available at www.jmp.com.